PYTHON

APRENDER PRACTICANDO

LUIS MINGUILLON PASCUAL

PYTHON

aprender practicando

Luis Minguillón Pascual

COPYRIGHT

La licencia de este libro electrónico es para uso personal. Por lo tanto no puedes revenderlo a otras personas.

Gracias por respetar los derechos de autor.

©Luis Minguillón Pascual
26/03/2026

Prefacio

En la actualidad, la programación se ha convertido en una habilidad esencial que trasciende profesiones y fronteras. Entre los muchos lenguajes disponibles, **Python** se ha destacado por su simplicidad, versatilidad y potencia, permitiendo desde la creación de aplicaciones web hasta el desarrollo de inteligencia artificial y análisis de datos. Su sintaxis clara y amigable lo hace ideal tanto para principiantes como para programadores experimentados que buscan resultados rápidos y eficientes.

Este libro, **“Python: Aprender Practicando”**, nace de la convicción de que la mejor manera de aprender a programar no es solo leyendo teoría, sino enfrentándose a retos concretos y construyendo proyectos paso a paso. La práctica constante permite no solo interiorizar conceptos, sino también desarrollar la creatividad, la lógica y la capacidad de resolver problemas de manera autónoma. Cada capítulo de este libro ha sido diseñado para acompañarte desde los fundamentos hasta la creación de proyectos completos, fomentando un aprendizaje activo y significativo.

No podría cerrar estas líneas sin expresar mi más profundo agradecimiento a mi esposa, **Marisa**, cuyo apoyo incondicional, paciencia y motivación han sido fundamentales durante todo el proceso de creación de este libro. Sin su aliento constante, la realización de este proyecto habría sido mucho más difícil.

Espero que, al recorrer estas páginas, encuentres no solo conocimientos técnicos, sino también la inspiración para explorar, experimentar y disfrutar del mundo de la programación en Python. Porque aprender practicando no solo construye habilidades, sino también confianza y pasión por crear.

Luis Minguillón Pascual

INDICE:

Prefacio

Indice

1. **Lección 1: Variables y Operadores en Python**
    - Conceptos, tipos de variables, asignación y reglas de nombrado.
    - Operadores aritméticos, de asignación, comparación, lógicos, identidad y pertenencia.
    - Conversión de tipos (casting) y buenas prácticas.
    - **Variables y Operadores: Programas Resueltos (1 a 100)**.
2. **Lección 2: Estructuras Condicionales en Python**
    - Instrucciones `if`, `if-else` e `if-elif-else`.
    - Condiciones booleanas, anidamiento y valores considerados `False`.
    - **Estructuras Condicionales: Programas Resueltos (1 a 40)**.
3. **Lección 3: Estructuras Repetitivas (Bucles) en Python**
    - Concepto de bucles y uso de `for` y `while`.
    - Función `range()`, bucles infinitos, `break`, `continue` y bucles anidados.
    - **Estructuras Repetitivas: BUCLES (Programas resueltos 1 a 40)**.
4. **Lección 4: Estructuras de Datos en Python – Listas (Lists)**
    - Creación, acceso, modificación y recorrido de listas.
    - Métodos principales: `append`, `remove`, `sort`, `reverse`, entre otros.
    - Slicing (rebanado) y List comprehension.
    - **Listas: Programas Resueltos (1 a 40)**.
5. **Lección 5: Estructuras de Datos en Python – Conjuntos (Sets)**
    - Características, creación y manipulación de elementos.
    - Operaciones matemáticas: unión, intersección, diferencia y diferencia simétrica.
    - Subconjuntos, superconjuntos y conjuntos inmutables (`frozenset`).
    - **Conjuntos: Programas Resueltos (1 a 50)**.
6. **Lección 6: Estructuras de Datos en Python – Tuplas (Tuples)**
    - Características, inmutabilidad y acceso mediante índices.
    - Funciones básicas, métodos y conversión entre listas y tuplas.
    - Desempaquetado de tuplas y aplicaciones prácticas.
    - **Tuplas: Programas Resueltos (1 a 50)**.
7. **Lección 7: Estructuras de Datos en Python – Diccionarios**
    - Estructura clave:valor, creación y acceso mediante el método `get()`.
    - Modificación, eliminación de elementos y métodos principales (`keys`, `values`, `items`).
    - Diccionarios anidados y comprensión de diccionarios.
    - **Diccionarios: Programas Resueltos (1 a 50)**.
8. **Lección 8: Funciones en Python**
    - Definición, parámetros y valor de retorno (`return`).

- Argumentos posicionales, nombrados, ámbito de variables (`scope`) y recursividad.
- Funciones lambda, parámetros arbitrarios (`*args`, `**kwargs`) y documentación.
- **Funciones: Programas Resueltos (1 a 50)**.

9. **Lección 9: Programación Orientada a Objetos (POO) en Python**
    - Conceptos fundamentales: Clase, Objeto, Instancia, Encapsulamiento, Herencia y Polimorfismo.
    - Método constructor `__init__`, atributos de instancia y de clase.
    - Métodos especiales (`__str__`, `__len__`, etc.), métodos estáticos y composición.
    - **POO: Programas Resueltos (1 a 50)**.
10. **Lección 10: Archivos y Directorios**
    - Apertura y cierre de archivos con `open()` y el contexto `with`.
    - Métodos de lectura y escritura, y manejo de errores con `try-except`.
    - Trabajo con rutas y directorios usando los módulos `os` y `pathlib`.
    - **Archivos y Directorios: Programas Resueltos (1 a 50)**.
11. **Lección 11: Archivos CSV**
    - Uso del módulo `csv` para crear, leer y modificar archivos tabulares.
    - Uso de `csv.DictReader` y `csv.DictWriter` para acceso por nombre de columna.
    - Ordenación de datos y uso de diferentes delimitadores.
    - **Archivos CSV: Programas Resueltos (1 a 50)**.
12. **Lección 12: Archivos JSON**
    - Introducción al formato JSON y funciones del módulo `json` (`dump`, `load`, `dumps`, `loads`).
    - Lectura, escritura y modificación de datos estructurados.
    - Buenas prácticas y trabajo con listas de objetos.
    - **Archivos JSON: Programas Resueltos (1 a 50)**.
13. **Lección 13: Formatos de Datos**
    - Formateo de números (notación científica, decimales) y f-strings.
    - Escapes de caracteres en cadenas y formato de fechas con `strftime`.
    - Resumen de formatos: JSON, CSV, Texto plano y Binario.
    - **Formatos de Datos: Programas Resueltos (1 a 50)**.
14. **Lección 14: Fechas y Horas**
    - Uso del módulo `datetime` para crear y obtener fechas y horas.
    - Operaciones con `timedelta` (sumar/restar tiempo) y comparaciones.
    - Uso de `calendar` y formatos locales con `locale`.
    - **Fechas y Horas: Programas Resueltos (1 a 50)**.

Epilogo

1. LECCIÓN: VARIABLES Y OPERADORES EN PYTHON

1. ¿Qué es una variable?

En Python, una **variable** es un espacio en la memoria que se utiliza para **almacenar información**. Cada variable tiene un **nombre único** y un **valor asociado**. Las variables permiten **guardar, modificar y utilizar datos** en el programa.

Ejemplo:
numero = 10
nombre = "Ana"
precio = 15.5

En este ejemplo:

- **numero** almacena un entero (**10**)
- **nombre** almacena una cadena de texto (**"Ana"**)
- **precio** almacena un número decimal (**15.5**)

2. Reglas para nombrar variables

- El nombre debe comenzar con una **letra** o un **guion bajo (_)**
- No puede comenzar con un número
- Solo puede contener **letras, números y guiones bajos**
- No puede ser una **palabra reservada de Python** como `for`, `if`, `while`, `def`, etc.
- Es recomendable que el nombre sea **descriptivo**, por ejemplo: edad_usuario, precio_total

3. Tipos de variables en Python

Python es un lenguaje **dinámicamente tipado**, lo que significa que no es necesario declarar el tipo de variable explícitamente; Python lo **detecta automáticamente** según el valor asignado.

Los tipos más comunes son:

- **int**: números enteros, ejemplo: 10, -5, 0
- **float**: números decimales, ejemplo: 3.14, -0.5
- **str**: cadenas de texto, ejemplo: "Hola", "Python"
- **bool**: valores lógicos, True o False
- **list**: listas de elementos, ejemplo: [1, 2, 3]
- **tuple**: tuplas, ejemplo: (1, 2, 3)
- **dict**: diccionarios, ejemplo: {"nombre": "Ana", "edad": 20}

4. Asignación de valores a variables

La **asignación** se realiza con el operador **=**.
Ejemplo:
x = 5
y = 10

```
suma = x + y
print(suma) # Imprime 15
```

También se puede **asignar el mismo valor a varias variables**:

```
a = b = c = 0
print(a, b, c) # Imprime 0 0 0
```

5. Operadores en Python

5.1 Operadores aritméticos

Sirven para realizar **operaciones matemáticas**.

- **+** : suma
- **-** : resta
- * : multiplicación
- **/** : división (devuelve float)
- **//** : división entera (devuelve solo la parte entera)
- **%** : módulo (devuelve el residuo)
- ** : potencia (exponente)

Ejemplo:

```
a = 10
b = 3
print(a + b) # 13
print(a - b) # 7
print(a * b) # 30
print(a / b) # 3.3333
print(a // b) # 3
print(a % b) # 1
print(a ** b) # 1000
```

5.2 Operadores de asignación

Permiten **modificar el valor de una variable** de manera abreviada.

- **=** : asignación simple
- **+=** : suma y asigna (x += 5 es x = x + 5)
- **-=** : resta y asigna
- ***=** : multiplica y asigna
- **/=** : divide y asigna
- **%=** : módulo y asigna
- **//=** : división entera y asigna
- ****=** : potencia y asigna

Ejemplo:
x = 10
x += 5 # ahora x = 15
x *= 2 # ahora x = 30

5.3 Operadores de comparación
Se usan para **comparar valores** y devuelven un valor booleano (**True o False**).

- == : igual a
- **!=** : diferente de
- > : mayor que
- < : menor que
- >= : mayor o igual que
- <= : menor o igual que

Ejemplo:
a = 10
b = 20
print(a == b) # False
print(a != b) # True
print(a < b) # True

5.4 Operadores lógicos
Permiten **combinar condiciones**.

- **and** : devuelve True si ambas condiciones son verdaderas
- **or** : devuelve True si alguna condición es verdadera
- **not** : invierte el valor booleano

Ejemplo:
edad = 18
sexo = "F"
print(edad >= 18 and sexo == "F") # True
print(edad < 18 or sexo == "F") # True
print(not(edad >= 18)) # False

5.5 Operadores de identidad
Se usan para **verificar si dos variables apuntan al mismo objeto**.

- **is** : True si ambos objetos son iguales
- **is not** : True si ambos objetos son diferentes

Ejemplo:
x = [1, 2, 3]
y = [1, 2, 3]
z = x

print(x is y) # False
print(x is z) # True

5.6 Operadores de pertenencia
Sirven para **verificar si un elemento está en una secuencia** (listas, cadenas, tuplas).

- **in** : True si el elemento está presente
- **not in** : True si el elemento no está presente

Ejemplo:
lista = [1, 2, 3, 4]
print(2 in lista) # True
print(5 not in lista) # True

6. Conversión de tipos de variables (casting)
A veces es necesario **convertir un tipo de dato a otro**.

- int(): convierte a entero
- float(): convierte a decimal
- str(): convierte a cadena
- bool(): convierte a booleano

Ejemplo:
x = "10"
y = int(x) + 5
print(y) # 15

7. Buenas prácticas con variables y operadores

- Usar nombres **claros y descriptivos**
- Evitar sobrescribir variables importantes sin necesidad
- Usar operadores de manera eficiente
- Evitar hacer operaciones entre tipos incompatibles sin conversión

8. Ejercicios prácticos sugeridos

1. Crear un programa que pida **dos números** y muestre su suma, resta, multiplicación y división.
2. Pedir al usuario su **edad** y determinar si puede votar.
3. Pedir un número y calcular su **cuadrado y cubo** usando operadores.
4. Pedir **tres números** y mostrar cuál es el mayor usando operadores de comparación.
5. Crear un programa que indique si un **número es par o impar** usando operadores módulo.

Esta lección cubre:

- **Concepto de variables**

- **Tipos de datos en Python**
- **Asignación y reglas de variables**
- **Todos los operadores importantes**
- **Conversión de tipos y buenas prácticas**
- **Ejemplos prácticos paso a paso**

1. VARIABLES Y OPERADORES. PROGRAMAS RESUELTOS

---

1. Sumar dos números ingresados por el usuario

```
# **Programa para sumar dos números**
numero1 = float(input("Ingrese el primer número: "))
numero2 = float(input("Ingrese el segundo número: "))

# **Suma**
suma = numero1 + numero2

print("La suma es:", suma)
```

---

2. Calcular el área de un rectángulo

```
# **Programa para calcular área de un rectángulo**
base = float(input("Ingrese la base del rectángulo: "))
altura = float(input("Ingrese la altura del rectángulo: "))

# **Cálculo del área**
area = base * altura

print("El área del rectángulo es:", area)
```

---

3. Convertir de grados Celsius a Fahrenheit

```
# **Programa para convertir Celsius a Fahrenheit**
celsius = float(input("Ingrese la temperatura en Celsius: "))

# **Conversión**
fahrenheit = (celsius * 9/5) + 32

print("La temperatura en Fahrenheit es:", fahrenheit)
```

---

4. Calcular el promedio de tres notas

```
# **Programa para calcular promedio**
nota1 = float(input("Ingrese la primera nota: "))
nota2 = float(input("Ingrese la segunda nota: "))
nota3 = float(input("Ingrese la tercera nota: "))

# **Cálculo del promedio**
promedio = (nota1 + nota2 + nota3) / 3

print("El promedio es:", promedio)
```

---

5. Determinar si un número es par o impar

```
# **Programa para saber si un número es par o impar**
numero = int(input("Ingrese un número: "))

# **Condición**
if numero % 2 == 0:
    print("El número es **par**")
else:
    print("El número es **impar**")
```

---

6. Calcular la edad de una persona

```
# **Programa para calcular edad**
anio_actual = int(input("Ingrese el año actual: "))
anio_nacimiento = int(input("Ingrese su año de nacimiento: "))

# **Cálculo de la edad**
edad = anio_actual - anio_nacimiento

print("Su edad es:", edad)
```

---

7. Calcular el área de un círculo

```
# **Programa para calcular área de un círculo**
import math

radio = float(input("Ingrese el radio del círculo: "))

# **Cálculo del área**
area = math.pi * radio ** 2

print("El área del círculo es:", area)
```

---

8. Calcular la hipotenusa de un triángulo rectángulo

```
# **Programa para calcular la hipotenusa**
import math

cateto1 = float(input("Ingrese el primer cateto: "))
cateto2 = float(input("Ingrese el segundo cateto: "))

# **Cálculo de la hipotenusa**
hipotenusa = math.sqrt(cateto1**2 + cateto2**2)

print("La hipotenusa es:", hipotenusa)
```

---

9. Convertir dólares a pesos

```
# **Programa para convertir dólares a pesos**
dolares = float(input("Ingrese la cantidad en dólares: "))
tasa_cambio = float(input("Ingrese la tasa de cambio actual: "))

# **Conversión**
pesos = dolares * tasa_cambio

print("Equivalente en pesos:", pesos)
```

---

10. Calcular el total de una compra con IVA

```
# **Programa para calcular total con IVA**
precio = float(input("Ingrese el precio del producto: "))
iva = float(input("Ingrese el porcentaje de IVA: "))

# **Cálculo del total**
total = precio + (precio * iva / 100)

print("El total a pagar es:", total)
```

---

11. Calcular la velocidad promedio

```
# **Programa para calcular velocidad promedio**
distancia = float(input("Ingrese la distancia recorrida (km): "))
tiempo = float(input("Ingrese el tiempo empleado (horas): "))

# **Cálculo de la velocidad promedio**
velocidad = distancia / tiempo

print("La velocidad promedio es:", velocidad, "km/h")
```

---

12. Calcular el área de un triángulo

```
# **Programa para calcular área de un triángulo**
base = float(input("Ingrese la base del triángulo: "))
altura = float(input("Ingrese la altura del triángulo: "))

# **Cálculo del área**
area = (base * altura) / 2

print("El área del triángulo es:", area)
```

---

13. Convertir minutos a horas y minutos

```
# **Programa para convertir minutos a horas y minutos**
minutos_totales = int(input("Ingrese la cantidad de minutos: "))

# **Cálculo**
horas = minutos_totales // 60
minutos = minutos_totales % 60

print(f"{minutos_totales} minutos son {horas} horas y {minutos} minutos")
```

14. Calcular el área de un cuadrado

```
# **Programa para calcular área de un cuadrado**
lado = float(input("Ingrese el lado del cuadrado: "))

# **Cálculo del área**
area = lado ** 2

print("El área del cuadrado es:", area)
```

---

15. Calcular el doble y el triple de un número

```
# **Programa para calcular doble y triple**
numero = float(input("Ingrese un número: "))

# **Cálculo**
doble = numero * 2
triple = numero * 3

print("El doble es:", doble)
print("El triple es:", triple)
```

---

16. Calcular el área de un trapecio

```
# **Programa para calcular área de un trapecio**
base_mayor = float(input("Ingrese la base mayor: "))
base_menor = float(input("Ingrese la base menor: "))
altura = float(input("Ingrese la altura: "))

# **Cálculo del área**
area = ((base_mayor + base_menor) * altura) / 2

print("El área del trapecio es:", area)
```

---

17. Convertir metros a kilómetros

```
# **Programa para convertir metros a kilómetros**
metros = float(input("Ingrese la cantidad de metros: "))

# **Conversión**
kilometros = metros / 1000

print(metros, "metros son", kilometros, "kilómetros")
```

---

18. Calcular el perímetro de un rectángulo

```
# **Programa para calcular perímetro de un rectángulo**
base = float(input("Ingrese la base: "))
altura = float(input("Ingrese la altura: "))

# **Cálculo del perímetro**
perimetro = 2 * (base + altura)

print("El perímetro del rectángulo es:", perimetro)
```

19. Calcular el área de un rombo

```
# **Programa para calcular área de un rombo**
diagonal_mayor = float(input("Ingrese la diagonal mayor: "))
diagonal_menor = float(input("Ingrese la diagonal menor: "))

# **Cálculo del área**
area = (diagonal_mayor * diagonal_menor) / 2

print("El área del rombo es:", area)
```

---

20. Determinar si un número es positivo, negativo o cero

```
# **Programa para determinar signo de un número**
numero = float(input("Ingrese un número: "))

# **Condiciones**
if numero > 0:
    print("El número es **positivo**")
elif numero < 0:
    print("El número es **negativo**")
else:
    print("El número es **cero**")
```

---

21. Calcular el perímetro de un círculo

```
# **Programa para calcular perímetro de un círculo**

import math


radio = float(input("Ingrese el radio del círculo: "))


# **Cálculo del perímetro (circunferencia)**

perimetro = 2 * math.pi * radio


print("El perímetro del círculo es:", perimetro)
```

---

22. Calcular el área de un pentágono regular

```
# **Programa para calcular área de un pentágono regular**

lado = float(input("Ingrese el lado del pentágono: "))

apotema = float(input("Ingrese el apotema: "))


# **Cálculo del área**

area = (5 * lado * apotema) / 2


print("El área del pentágono es:", area)
```

23. Calcular el promedio ponderado de tres notas

```
# **Programa para promedio ponderado**
nota1 = float(input("Ingrese la primera nota: "))
peso1 = float(input("Ingrese el peso de la primera nota (%): "))
nota2 = float(input("Ingrese la segunda nota: "))
peso2 = float(input("Ingrese el peso de la segunda nota (%): "))
nota3 = float(input("Ingrese la tercera nota: "))
peso3 = float(input("Ingrese el peso de la tercera nota (%): "))

# **Cálculo del promedio ponderado**
promedio = (nota1*peso1 + nota2*peso2 + nota3*peso3) / (peso1 + peso2 + peso3)

print("El promedio ponderado es:", promedio)
```

---

24. Calcular el interés simple

```
# **Programa para calcular interés simple**
capital = float(input("Ingrese el capital: "))
tasa = float(input("Ingrese la tasa de interés (% anual): "))
tiempo = float(input("Ingrese el tiempo en años: "))

# **Cálculo del interés simple**
interes = (capital * tasa * tiempo) / 100

print("El interés simple es:", interes)
```

---

25. Calcular el interés compuesto

```
# **Programa para calcular interés compuesto**
capital = float(input("Ingrese el capital: "))
tasa = float(input("Ingrese la tasa de interés (% anual): "))
tiempo = float(input("Ingrese el tiempo en años: "))

# **Cálculo del monto final**
monto = capital * (1 + tasa/100) ** tiempo

print("El monto final con interés compuesto es:", monto)
```

---

26. Calcular la suma de los primeros n números enteros

```
# **Programa para sumar los primeros n números**
n = int(input("Ingrese un número entero positivo: "))

# **Cálculo de la suma**
suma = n * (n + 1) // 2

print("La suma de los primeros", n, "números enteros es:", suma)
```

---

27. Calcular el factorial de un número

```
# **Programa para calcular factorial**
numero = int(input("Ingrese un número entero: "))
factorial = 1

# **Cálculo del factorial**
for i in range(1, numero + 1):
    factorial *= i

print("El factorial de", numero, "es:", factorial)
```

---

28. Convertir segundos a horas, minutos y segundos

```
# **Programa para convertir segundos a horas, minutos y segundos**
segundos_totales = int(input("Ingrese los segundos: "))

# **Cálculos**
horas = segundos_totales // 3600
minutos = (segundos_totales % 3600) // 60
segundos = segundos_totales % 60

print(f"{segundos_totales} segundos son {horas} horas, {minutos} minutos y
{segundos} segundos")
```

---

29. Determinar si un año es bisiesto

```
# **Programa para determinar si un año es bisiesto**
anio = int(input("Ingrese un año: "))

# **Condición**
if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):
    print("El año es **bisiesto**")
else:
    print("El año no es bisiesto")
```

---

30. Calcular el valor absoluto de un número

```
# **Programa para calcular valor absoluto**
numero = float(input("Ingrese un número: "))

# **Cálculo**
valor_absoluto = abs(numero)

print("El valor absoluto es:", valor_absoluto)
```

---

31. Calcular el promedio de n números ingresados por el usuario

```
# **Programa para calcular promedio de n números**
n = int(input("¿Cuántos números desea ingresar? "))
suma = 0

# **Entrada de números y suma**
for i in range(n):
    numero = float(input(f"Ingrese el número {i+1}: "))
    suma += numero

# **Cálculo del promedio**
promedio = suma / n

print("El promedio es:", promedio)
```

---

32. Determinar el mayor de tres números

```
# **Programa para determinar el mayor de tres números**
num1 = float(input("Ingrese el primer número: "))
num2 = float(input("Ingrese el segundo número: "))
num3 = float(input("Ingrese el tercer número: "))

# **Comparación**
mayor = num1
if num2 > mayor:
    mayor = num2
if num3 > mayor:
    mayor = num3

print("El número mayor es:", mayor)
```

---

33. Determinar el menor de tres números

```
# **Programa para determinar el menor de tres números**
num1 = float(input("Ingrese el primer número: "))
num2 = float(input("Ingrese el segundo número: "))
num3 = float(input("Ingrese el tercer número: "))

# **Comparación**
menor = num1
if num2 < menor:
    menor = num2
if num3 < menor:
    menor = num3

print("El número menor es:", menor)
```

---

34. Generar la tabla de multiplicar de un número

```
# **Programa para generar tabla de multiplicar**
numero = int(input("Ingrese un número: "))

# **Tabla de multiplicar**
for i in range(1, 11):
    print(f"{numero} x {i} = {numero * i}")
```

---

35. Calcular el área de un cilindro

```
# **Programa para calcular área de un cilindro**
import math

radio = float(input("Ingrese el radio del cilindro: "))
altura = float(input("Ingrese la altura del cilindro: "))

# **Cálculo del área**
area = 2 * math.pi * radio * (radio + altura)

print("El área del cilindro es:", area)
```

---

36. Calcular el volumen de un cilindro

```
# **Programa para calcular volumen de un cilindro**
import math

radio = float(input("Ingrese el radio del cilindro: "))
altura = float(input("Ingrese la altura del cilindro: "))

# **Cálculo del volumen**
volumen = math.pi * radio**2 * altura

print("El volumen del cilindro es:", volumen)
```

---

37. Calcular el perímetro de un triángulo

```
# **Programa para calcular perímetro de un triángulo**
lado1 = float(input("Ingrese el primer lado: "))
lado2 = float(input("Ingrese el segundo lado: "))
lado3 = float(input("Ingrese el tercer lado: "))

# **Cálculo del perímetro**
perimetro = lado1 + lado2 + lado3

print("El perímetro del triángulo es:", perimetro)
```

---

38. Calcular el área de un elipse

```
# **Programa para calcular área de una elipse**
import math

eje_mayor = float(input("Ingrese el eje mayor: "))
eje_menor = float(input("Ingrese el eje menor: "))

# **Cálculo del área**
area = math.pi * eje_mayor * eje_menor

print("El área de la elipse es:", area)
```

---

39. Convertir libras a kilogramos

```
# **Programa para convertir libras a kilogramos**
libras = float(input("Ingrese el peso en libras: "))

# **Conversión**
kilogramos = libras * 0.453592

print(libras, "libras son", kilogramos, "kilogramos")
```

---

40. Convertir kilómetros a millas

```
# **Programa para convertir kilómetros a millas**
kilometros = float(input("Ingrese los kilómetros: "))

# **Conversión**
millas = kilometros * 0.621371

print(kilometros, "kilómetros son", millas, "millas")
```

---

41. Calcular la suma de los dígitos de un número

```
# **Programa para sumar los dígitos de un número**
numero = int(input("Ingrese un número entero: "))
suma = 0
temp = numero

# **Suma de dígitos**
while temp > 0:
    digito = temp % 10
    suma += digito
    temp = temp // 10

print("La suma de los dígitos de", numero, "es:", suma)
```

---

42. Invertir un número

```
# **Programa para invertir un número**
numero = int(input("Ingrese un número entero: "))
invertido = 0
temp = numero

# **Invertir número**
while temp > 0:
    digito = temp % 10
    invertido = invertido * 10 + digito
    temp = temp // 10

print("El número invertido es:", invertido)
```

---

43. Determinar si un número es primo

```
# **Programa para determinar si un número es primo**
numero = int(input("Ingrese un número entero mayor que 1: "))

# **Verificación de primo**
if numero <= 1:
    print("El número no es primo")
else:
    es_primo = True
    for i in range(2, int(numero**0.5) + 1):
        if numero % i == 0:
            es_primo = False
            break
    if es_primo:
        print("El número es **primo**")
    else:
        print("El número no es primo")
```

---

44. Generar los n primeros números primos

```
# **Programa para generar los n primeros números primos**
n = int(input("Ingrese la cantidad de números primos a generar: "))
contador = 0
numero = 2

# **Generación de primos**
while contador < n:
    es_primo = True
    for i in range(2, int(numero**0.5) + 1):
        if numero % i == 0:
            es_primo = False
            break
    if es_primo:
        print(numero, end=" ")
        contador += 1
    numero += 1
print()
```

45. Calcular la suma de los números pares hasta n

```
# **Programa para sumar números pares hasta n**
n = int(input("Ingrese un número entero positivo: "))
suma = 0

# **Suma de pares**
for i in range(2, n+1, 2):
    suma += i

print("La suma de los números pares hasta", n, "es:", suma)
```

---

46. Calcular la suma de los números impares hasta n

```
# **Programa para sumar números impares hasta n**
n = int(input("Ingrese un número entero positivo: "))
suma = 0

# **Suma de impares**
for i in range(1, n+1, 2):
    suma += i

print("La suma de los números impares hasta", n, "es:", suma)
```

---

47. Contar la cantidad de dígitos de un número

```
# **Programa para contar dígitos**
numero = int(input("Ingrese un número entero: "))
contador = 0
temp = numero

# **Conteo de dígitos**
while temp > 0:
    temp = temp // 10
    contador += 1

print("El número", numero, "tiene", contador, "dígitos")
```

---

48. Verificar si un número es capicúa

```
# **Programa para verificar si un número es capicúa**
numero = int(input("Ingrese un número entero: "))
invertido = 0
temp = numero

# **Invertir número**
while temp > 0:
    digito = temp % 10
    invertido = invertido * 10 + digito
    temp = temp // 10

# **Comparación**
if numero == invertido:
    print("El número es **capicúa**")
else:
    print("El número no es capicúa")
```

---

49. Calcular la media aritmética de n números

```
# **Programa para calcular media aritmética**
n = int(input("Ingrese la cantidad de números: "))
suma = 0

# **Ingreso de números**
for i in range(n):
    numero = float(input(f"Ingrese el número {i+1}: "))
    suma += numero

media = suma / n

print("La media aritmética es:", media)
```

---

50. Calcular el promedio de n números positivos

```
# **Programa para calcular promedio de números positivos**
n = int(input("¿Cuántos números positivos desea ingresar? "))
suma = 0
contador = 0

# **Ingreso y suma**
for i in range(n):
    numero = float(input(f"Ingrese el número {i+1}: "))
    if numero >= 0:
        suma += numero
        contador += 1

if contador > 0:
    promedio = suma / contador
    print("El promedio de los números positivos es:", promedio)
else:
    print("No se ingresaron números positivos")
```

---

51. Calcular el área de un cuadrado a partir de su perímetro

```
# **Programa para calcular área de un cuadrado a partir del perímetro**
perimetro = float(input("Ingrese el perímetro del cuadrado: "))

# **Cálculo del lado**
lado = perimetro / 4

# **Cálculo del área**
area = lado ** 2

print("El área del cuadrado es:", area)
```

---

52. Calcular el área de un triángulo a partir de sus lados (Fórmula de Herón)

```
# **Programa para calcular área de un triángulo usando fórmula de Herón**
a = float(input("Ingrese el primer lado: "))
b = float(input("Ingrese el segundo lado: "))
c = float(input("Ingrese el tercer lado: "))

# **Semiperímetro**
s = (a + b + c) / 2

# **Cálculo del área**
import math
area = math.sqrt(s * (s - a) * (s - b) * (s - c))

print("El área del triángulo es:", area)
```

---

53. Calcular el total a pagar con descuento

```
# **Programa para calcular total con descuento**
precio = float(input("Ingrese el precio del producto: "))
descuento = float(input("Ingrese el porcentaje de descuento: "))

# **Cálculo del total**
total = precio - (precio * descuento / 100)

print("El total a pagar es:", total)
```

---

54. Convertir metros cuadrados a hectáreas

```
# **Programa para convertir metros cuadrados a hectáreas**
metros_cuadrados = float(input("Ingrese los metros cuadrados: "))

# **Conversión**
hectareas = metros_cuadrados / 10000

print(metros_cuadrados, "m² son", hectareas, "hectáreas")
```

---

55. Calcular la velocidad con distancia y tiempo en minutos

```
# **Programa para calcular velocidad con tiempo en minutos**
distancia = float(input("Ingrese la distancia recorrida (km): "))
tiempo_min = float(input("Ingrese el tiempo en minutos: "))

# **Conversión de tiempo a horas**
tiempo_horas = tiempo_min / 60

# **Cálculo de velocidad**
velocidad = distancia / tiempo_horas

print("La velocidad promedio es:", velocidad, "km/h")
```

---

56. Calcular el área de un romboide

```
# **Programa para calcular área de un romboide**
base = float(input("Ingrese la base: "))
altura = float(input("Ingrese la altura: "))

# **Cálculo del área**
area = base * altura

print("El área del romboide es:", area)
```

---

57. Calcular el perímetro de un pentágono regular

```
# **Programa para calcular perímetro de un pentágono regular**
lado = float(input("Ingrese la longitud del lado: "))

# **Cálculo del perímetro**
perimetro = 5 * lado

print("El perímetro del pentágono es:", perimetro)
```

---

58. Calcular el promedio de n calificaciones con validación de 0 a 100

```
# **Programa para calcular promedio con validación**
n = int(input("Ingrese la cantidad de calificaciones: "))
suma = 0

for i in range(n):
    nota = float(input(f"Ingrese la calificación {i+1} (0-100): "))
    while nota < 0 or nota > 100:
        print("Calificación inválida. Debe estar entre 0 y 100.")
        nota = float(input(f"Ingrese la calificación {i+1} (0-100): "))
    suma += nota

promedio = suma / n
print("El promedio es:", promedio)
```

---

59. Convertir kilómetros por hora a metros por segundo

```
# **Programa para convertir km/h a m/s**
velocidad_kmh = float(input("Ingrese la velocidad en km/h: "))

# **Conversión**
velocidad_ms = velocidad_kmh / 3.6

print(velocidad_kmh, "km/h son", velocidad_ms, "m/s")
```

---

60. Convertir metros por segundo a kilómetros por hora

```
# **Programa para convertir m/s a km/h**
velocidad_ms = float(input("Ingrese la velocidad en m/s: "))

# **Conversión**
velocidad_kmh = velocidad_ms * 3.6

print(velocidad_ms, "m/s son", velocidad_kmh, "km/h")
```

---

61. Calcular el área de un hexágono regular

```
# **Programa para calcular área de un hexágono regular**
lado = float(input("Ingrese la longitud del lado del hexágono: "))

# **Cálculo del área**
import math
area = (3 * math.sqrt(3) * lado**2) / 2

print("El área del hexágono es:", area)
```

---

62. Calcular el índice de masa corporal (IMC)

```
# **Programa para calcular IMC**
peso = float(input("Ingrese su peso en kg: "))
altura = float(input("Ingrese su altura en metros: "))

# **Cálculo del IMC**
imc = peso / (altura ** 2)

print("Su índice de masa corporal (IMC) es:", imc)
```

---

63. Determinar la categoría según el IMC

```
# **Programa para categorizar según IMC**
imc = float(input("Ingrese su IMC: "))

# **Condiciones**
if imc < 18.5:
    print("Bajo peso")
elif 18.5 <= imc < 25:
    print("Normal")
elif 25 <= imc < 30:
    print("Sobrepeso")
else:
    print("Obesidad")
```

---

64. Calcular la edad en meses

```
# **Programa para calcular edad en meses**
edad_anios = int(input("Ingrese su edad en años: "))

# **Cálculo de edad en meses**
edad_meses = edad_anios * 12

print("Su edad en meses es:", edad_meses)
```

---

65. Calcular la suma de los cuadrados de n números

```
# **Programa para sumar cuadrados de n números**
n = int(input("Ingrese la cantidad de números: "))
suma = 0

for i in range(n):
    numero = float(input(f"Ingrese el número {i+1}: "))
    suma += numero**2

print("La suma de los cuadrados es:", suma)
```

---

66. Calcular el promedio de n números pares

```
# **Programa para calcular promedio de números pares**
n = int(input("¿Cuántos números pares desea ingresar? "))
suma = 0
contador = 0

for i in range(n):
    numero = int(input(f"Ingrese el número {i+1}: "))
    if numero % 2 == 0:
        suma += numero
        contador += 1

if contador > 0:
    promedio = suma / contador
    print("El promedio de los números pares es:", promedio)
else:
    print("No se ingresaron números pares")
```

---

67. Determinar si un número es divisible entre 3 y 5

```
# **Programa para verificar divisibilidad entre 3 y 5**
numero = int(input("Ingrese un número: "))

# **Condición**
if numero % 3 == 0 and numero % 5 == 0:
    print("El número es divisible entre 3 y 5")
else:
    print("El número no es divisible entre 3 y 5")
```

---

68. Calcular la suma de los números múltiplos de 3 hasta n

```
# **Programa para sumar múltiplos de 3 hasta n**
n = int(input("Ingrese un número entero positivo: "))
suma = 0

for i in range(3, n+1, 3):
    suma += i

print("La suma de los múltiplos de 3 hasta", n, "es:", suma)
```

---

69. Calcular la suma de los números múltiplos de 5 hasta n

```
# **Programa para sumar múltiplos de 5 hasta n**
n = int(input("Ingrese un número entero positivo: "))
suma = 0

for i in range(5, n+1, 5):
    suma += i

print("La suma de los múltiplos de 5 hasta", n, "es:", suma)
```

---

70. Determinar si un número es divisible entre 2, 3 o 5

```
# **Programa para verificar divisibilidad entre 2, 3 o 5**
numero = int(input("Ingrese un número: "))

# **Condición**
if numero % 2 == 0:
    print("El número es divisible entre 2")
if numero % 3 == 0:
    print("El número es divisible entre 3")
if numero % 5 == 0:
    print("El número es divisible entre 5")
if numero % 2 != 0 and numero % 3 != 0 and numero % 5 != 0:
    print("El número no es divisible entre 2, 3 ni 5")
```

---

71. Calcular el área de un triángulo equilátero

```
# **Programa para calcular área de un triángulo equilátero**
lado = float(input("Ingrese la longitud del lado: "))

# **Cálculo del área**
import math
area = (math.sqrt(3) / 4) * lado ** 2

print("El área del triángulo equilátero es:", area)
```

---

72. Calcular el área de un trapecio usando bases y altura

```
# **Programa para calcular área de un trapecio**
base_mayor = float(input("Ingrese la base mayor: "))
base_menor = float(input("Ingrese la base menor: "))
altura = float(input("Ingrese la altura: "))

# **Cálculo del área**
area = ((base_mayor + base_menor) * altura) / 2

print("El área del trapecio es:", area)
```

---

73. Calcular el perímetro de un círculo a partir del diámetro

```
# **Programa para calcular perímetro de un círculo a partir del diámetro**
diametro = float(input("Ingrese el diámetro del círculo: "))

# **Cálculo del perímetro**
import math
perimetro = math.pi * diametro

print("El perímetro del círculo es:", perimetro)
```

---

74. Calcular el área de un triángulo a partir de base y altura

```
# **Programa para calcular área de un triángulo**
base = float(input("Ingrese la base del triángulo: "))
altura = float(input("Ingrese la altura del triángulo: "))

# **Cálculo del área**
area = (base * altura) / 2

print("El área del triángulo es:", area)
```

---

75. Calcular el volumen de una esfera

```
# **Programa para calcular volumen de una esfera**
import math

radio = float(input("Ingrese el radio de la esfera: "))

# **Cálculo del volumen**
volumen = (4/3) * math.pi * radio ** 3

print("El volumen de la esfera es:", volumen)
```

---

76. Calcular el área de una esfera

```
# **Programa para calcular área de una esfera**
import math

radio = float(input("Ingrese el radio de la esfera: "))

# **Cálculo del área**
area = 4 * math.pi * radio ** 2

print("El área de la esfera es:", area)
```

---

77. Calcular la suma de los números múltiplos de 7 hasta n

```
# **Programa para sumar múltiplos de 7 hasta n**
n = int(input("Ingrese un número entero positivo: "))
suma = 0

for i in range(7, n+1, 7):
    suma += i

print("La suma de los múltiplos de 7 hasta", n, "es:", suma)
```

---

78. Determinar si un número es divisible entre 4 y 6

```
# **Programa para verificar divisibilidad entre 4 y 6**
numero = int(input("Ingrese un número: "))

# **Condición**
if numero % 4 == 0 and numero % 6 == 0:
    print("El número es divisible entre 4 y 6")
else:
    print("El número no es divisible entre 4 y 6")
```

---

79. Calcular el área de un círculo a partir del perímetro

```
# **Programa para calcular área de un círculo a partir del perímetro**
import math

perimetro = float(input("Ingrese el perímetro del círculo: "))

# **Cálculo del radio**
radio = perimetro / (2 * math.pi)

# **Cálculo del área**
area = math.pi * radio ** 2

print("El área del círculo es:", area)
```

---

80. Calcular el perímetro de un rectángulo a partir del área y un lado

```
# **Programa para calcular perímetro de un rectángulo a partir del área y un lado**

area = float(input("Ingrese el área del rectángulo: "))

lado = float(input("Ingrese un lado del rectángulo: "))


# **Cálculo del otro lado**

otro_lado = area / lado


# **Cálculo del perímetro**

perimetro = 2 * (lado + otro_lado)


print("El perímetro del rectángulo es:", perimetro)
```

¡Perfecto! Aquí van los siguientes **10 programas (81-90)**:

---

81. Calcular la raíz cuadrada de un número

```
# **Programa para calcular raíz cuadrada**

import math


numero = float(input("Ingrese un número: "))


# **Cálculo de la raíz cuadrada**

raiz = math.sqrt(numero)


print("La raíz cuadrada de", numero, "es:", raiz)
```

---

82. Calcular la potencia de un número

```
# **Programa para calcular potencia**

base = float(input("Ingrese la base: "))

exponente = float(input("Ingrese el exponente: "))


# **Cálculo de la potencia**

resultado = base ** exponente


print(f"{base} elevado a {exponente} es:", resultado)
```

---

83. Calcular el área de un cono

```
# **Programa para calcular área de un cono**
import math

radio = float(input("Ingrese el radio del cono: "))
altura = float(input("Ingrese la altura del cono: "))

# **Cálculo de la generatriz**
generatriz = math.sqrt(radio**2 + altura**2)

# **Cálculo del área**
area = math.pi * radio * (radio + generatriz)

print("El área del cono es:", area)
```

---

84. Calcular el volumen de un cono

```
# **Programa para calcular volumen de un cono**
import math

radio = float(input("Ingrese el radio del cono: "))
altura = float(input("Ingrese la altura del cono: "))

# **Cálculo del volumen**
volumen = (1/3) * math.pi * radio**2 * altura

print("El volumen del cono es:", volumen)
```

---

85. Calcular el área de un prisma rectangular

```
# **Programa para calcular área de un prisma rectangular**
largo = float(input("Ingrese el largo: "))
ancho = float(input("Ingrese el ancho: "))
alto = float(input("Ingrese la altura: "))

# **Cálculo del área**
area = 2 * (largo*ancho + largo*alto + ancho*alto)

print("El área del prisma rectangular es:", area)
```

---

86. Calcular el volumen de un prisma rectangular

```
# **Programa para calcular volumen de un prisma rectangular**
largo = float(input("Ingrese el largo: "))
ancho = float(input("Ingrese el ancho: "))
alto = float(input("Ingrese la altura: "))

# **Cálculo del volumen**
volumen = largo * ancho * alto

print("El volumen del prisma rectangular es:", volumen)
```

---

87. Calcular el promedio de n números negativos

```
# **Programa para calcular promedio de números negativos**
n = int(input("Ingrese la cantidad de números: "))
suma = 0
contador = 0

for i in range(n):
    numero = float(input(f"Ingrese el número {i+1}: "))
    if numero < 0:
        suma += numero
        contador += 1

if contador > 0:
    promedio = suma / contador
    print("El promedio de los números negativos es:", promedio)
else:
    print("No se ingresaron números negativos")
```

---

88. Verificar si un número es divisible entre 7 o 11

```
# **Programa para verificar divisibilidad entre 7 o 11**
numero = int(input("Ingrese un número: "))

# **Condición**
if numero % 7 == 0 or numero % 11 == 0:
    print("El número es divisible entre 7 o 11")
else:
    print("El número no es divisible entre 7 ni 11")
```

---

89. Calcular la suma de los números múltiplos de 2, 3 y 5 hasta n

```
# **Programa para sumar múltiplos de 2, 3 y 5 hasta n**
n = int(input("Ingrese un número entero positivo: "))
suma = 0

for i in range(1, n+1):
    if i % 2 == 0 or i % 3 == 0 or i % 5 == 0:
        suma += i

print("La suma de los múltiplos de 2, 3 y 5 hasta", n, "es:", suma)
```

---

90. Calcular la suma de los primeros n números impares

```
# **Programa para sumar los primeros n números impares**
n = int(input("Ingrese la cantidad de números impares a sumar: "))
suma = 0
contador = 0
numero = 1

# **Suma de impares**
while contador < n:
    suma += numero
    numero += 2
    contador += 1

print("La suma de los primeros", n, "números impares es:", suma)
```

---

91. Calcular el área de un cuadrado a partir de la diagonal

```
# **Programa para calcular área de un cuadrado usando diagonal**
diagonal = float(input("Ingrese la diagonal del cuadrado: "))

# **Cálculo del lado**
lado = diagonal / (2**0.5)

# **Cálculo del área**
area = lado ** 2

print("El área del cuadrado es:", area)
```

---

92. Calcular el área de un triángulo a partir de los tres lados (Herón)

```
# **Programa para calcular área de un triángulo usando fórmula de Herón**
a = float(input("Ingrese el primer lado: "))
b = float(input("Ingrese el segundo lado: "))
c = float(input("Ingrese el tercer lado: "))

# **Semiperímetro**
s = (a + b + c) / 2

# **Cálculo del área**
import math
area = math.sqrt(s * (s - a) * (s - b) * (s - c))

print("El área del triángulo es:", area)
```

---

93. Calcular el volumen de una pirámide

```
# **Programa para calcular volumen de una pirámide**
base = float(input("Ingrese el área de la base: "))
altura = float(input("Ingrese la altura de la pirámide: "))

# **Cálculo del volumen**
volumen = (1/3) * base * altura

print("El volumen de la pirámide es:", volumen)
```

---

94. Calcular el área de un círculo a partir del radio

```
# **Programa para calcular área de un círculo**
import math

radio = float(input("Ingrese el radio del círculo: "))

# **Cálculo del área**
area = math.pi * radio ** 2

print("El área del círculo es:", area)
```

---

95. Convertir grados Fahrenheit a Celsius

```
# **Programa para convertir Fahrenheit a Celsius**
fahrenheit = float(input("Ingrese la temperatura en Fahrenheit: "))

# **Conversión**
celsius = (fahrenheit - 32) * 5 / 9

print(fahrenheit, "°F son", celsius, "°C")
```

---

96. Calcular el promedio de n números con decimales

```
# **Programa para calcular promedio de n números**
n = int(input("Ingrese la cantidad de números: "))
suma = 0.0

for i in range(n):
    numero = float(input(f"Ingrese el número {i+1}: "))
    suma += numero

promedio = suma / n

print("El promedio es:", promedio)
```

---

97. Calcular el perímetro de un triángulo equilátero

```
# **Programa para calcular perímetro de un triángulo equilátero**
lado = float(input("Ingrese la longitud del lado: "))

# **Cálculo del perímetro**
perimetro = 3 * lado

print("El perímetro del triángulo equilátero es:", perimetro)
```

---

98. Calcular el área de un cuadrado a partir del perímetro

```
# **Programa para calcular área de un cuadrado usando perímetro**
perimetro = float(input("Ingrese el perímetro del cuadrado: "))

# **Cálculo del lado**
lado = perimetro / 4

# **Cálculo del área**
area = lado ** 2

print("El área del cuadrado es:", area)
```

---

99. Calcular el área de un rectángulo a partir del perímetro y un lado

```
# **Programa para calcular área de un rectángulo a partir de perímetro y un lado**
perimetro = float(input("Ingrese el perímetro del rectángulo: "))
lado1 = float(input("Ingrese un lado del rectángulo: "))

# **Cálculo del otro lado**
lado2 = (perimetro / 2) - lado1

# **Cálculo del área**
area = lado1 * lado2

print("El área del rectángulo es:", area)
```

---

100. Calcular el área de un círculo a partir del diámetro

```
# **Programa para calcular área de un círculo usando diámetro**
import math

diametro = float(input("Ingrese el diámetro del círculo: "))

# **Cálculo del radio**
radio = diametro / 2

# **Cálculo del área**
area = math.pi * radio ** 2

print("El área del círculo es:", area)
```

---

2. LECCIÓN: ESTRUCTURAS CONDICIONALES EN PYTHON

1. ¿Qué son las estructuras condicionales?

Las **estructuras condicionales** permiten que un programa **tome decisiones** según ciertas condiciones. Esto significa que el flujo del programa puede cambiar dependiendo de si una condición es **verdadera (True)** o **falsa (False)**.

2. La instrucción if

La instrucción **if** se utiliza para ejecutar un bloque de código **solo si se cumple una condición**.

Ejemplo:

```
numero = 10
if numero > 0:
print("El número es positivo")
```

En este ejemplo:

- La condición es **numero > 0**
- El bloque indentado bajo **if** se ejecuta solo si la condición es verdadera

3. La instrucción if-else

La instrucción **if-else** permite **ejecutar un bloque si la condición es verdadera y otro bloque si es falsa**.

Ejemplo:

```
numero = -5
if numero > 0:
print("El número es positivo")
else:
print("El número es cero o negativo")
```

4. La instrucción if-elif-else

Se utiliza cuando hay **múltiples condiciones posibles**. La estructura es:

```
if condición1:
# bloque1
elif condición2:
# bloque2
elif condición3:
# bloque3
else:
# bloque por defecto
```

Ejemplo:

```
nota = 85
if nota >= 90:
print("Excelente")
elif nota >= 75:
print("Bueno")
elif nota >= 60:
print("Regular")
```

else:
print("Deficiente")

5. Condiciones booleanas
Las condiciones en Python devuelven un valor **booleano (True o False)**. Pueden utilizarse:

- Operadores de comparación: ==, !=, >, <, >=, <=
- Operadores lógicos: and, or, not

Ejemplo:
edad = 20
sexo = "M"
if edad >= 18 and sexo == "M":
print("Puede aplicar al registro masculino adulto")

6. Operadores de comparación en condicionales

- **==** : igual a
- **!=** : diferente de
- **>** : mayor que
- **<** : menor que
- **>=** : mayor o igual que
- **<=** : menor o igual que

Ejemplo:
a = 10
b = 20
if a < b:
print("a es menor que b")

7. Operadores lógicos en condicionales

- **and** : devuelve True si **todas las condiciones** son verdaderas
- **or** : devuelve True si **alguna condición** es verdadera
- **not** : invierte el valor de la condición

Ejemplo:
numero = 15
if numero > 10 and numero < 20:
print("El número está entre 10 y 20")

if numero < 10 or numero > 20:
print("El número está fuera del rango 10-20")

if not(numero == 15):
print("El número no es 15")
else:
print("El número es 15")

8. Anidamiento de condicionales
Es posible colocar **un if dentro de otro if**, lo que permite decisiones más complejas.

Ejemplo:
edad = 25
sexo = "F"
if edad >= 18:
if sexo == "F":
print("Mayor de edad y mujer")
else:
print("Mayor de edad y hombre")
else:
print("Menor de edad")

9. Condiciones combinadas
Las condiciones pueden combinarse de múltiples formas usando paréntesis para **controlar la prioridad**.

Ejemplo:
numero = 12
if (numero > 10 and numero < 20) or numero == 25:
print("Cumple la condición")
else:
print("No cumple la condición")

10. Valores considerados como False en Python
En Python, los siguientes valores se consideran **falsos en condicionales**:

- False
- 0
- 0.0
- "" (cadena vacía)
- [] (lista vacía)
- {} (diccionario vacío)
- None

Ejemplo:
lista = []
if not lista:
print("La lista está vacía")

11. Uso de condicionales con cadenas
Se pueden comparar cadenas usando operadores de comparación y verificar contenido con **in** o **not in**.

Ejemplo:
nombre = "Ana"

```
if nombre == "Ana":
print("Hola Ana")

if "a" in nombre:
print("Contiene la letra a")
```

12. Ejemplos prácticos de condicionales

1. Verificar si un número es positivo, negativo o cero
   ```
   numero = float(input("Ingresa un número: "))
   if numero > 0:
   print("Positivo")
   elif numero < 0:
   print("Negativo")
   else:
   print("Cero")
   ```
2. Determinar si un estudiante aprueba o reprueba según nota
   ```
   nota = float(input("Ingresa la nota: "))
   if nota >= 60:
   print("Aprobado")
   else:
   print("Reprobado")
   ```
3. Clasificar edad
   ```
   edad = int(input("Edad: "))
   if edad < 13:
   print("Niño")
   elif edad <= 19:
   print("Adolescente")
   elif edad <= 59:
   print("Adulto")
   else:
   print("Adulto mayor")
   ```
4. Verificar acceso por contraseña
   ```
   contraseña = input("Ingresa contraseña: ")
   if contraseña == "1234":
   print("Acceso permitido")
   else:
   print("Acceso denegado")
   ```
5. Determinar si un año es bisiesto
   ```
   anio = int(input("Año: "))
   if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):
   print("Bisiesto")
   else:
   print("No es bisiesto")
   ```

13. Buenas prácticas al usar condicionales

- Indentar correctamente el código
- Usar nombres de variables descriptivos
- Evitar condicionales muy largos; se pueden **dividir en varias líneas**
- Usar **elif** en lugar de múltiples if independientes para eficiencia
- Comentar el código cuando la condición es compleja

14. Resumen de estructuras condicionales

- **if**: ejecutar un bloque si la condición es verdadera
- **if-else**: ejecutar un bloque si es verdadera y otro si es falsa
- **if-elif-else**: manejar múltiples condiciones
- **Condicionales anidadas**: if dentro de if para decisiones complejas
- **Operadores de comparación y lógicos**: controlar las condiciones de manera precisa
- **Valores falsos**: recordar qué valores Python considera False

15. Ejercicios sugeridos para practicar

1. Pedir un número e indicar si es **par o impar**.
2. Pedir tres números e indicar cuál es el **mayor**.
3. Pedir edad y mostrar un mensaje según rango: niño, adolescente, adulto, adulto mayor.
4. Pedir contraseña hasta que sea correcta.
5. Verificar si un número está dentro de un rango específico usando **operadores lógicos**.
6. Pedir tres notas y determinar si el estudiante **aprueba, está en recuperación o reprueba**.

Esta lección cubre:

- **Concepto de estructuras condicionales**
- **Tipos de condicionales (if, if-else, if-elif-else)**
- **Condiciones booleanas y operadores**
- **Condicionales anidadas y combinadas**
- **Condicionales aplicadas a números, cadenas y listas**
- **Ejemplos prácticos paso a paso**
- **Buenas prácticas y ejercicios para practicar**

2. ESTRUCTURAS CONDICIONALES

1. Verificar si un número es positivo, negativo o cero

```
numero = float(input("Ingresa un número: "))
if numero > 0:
    print("El número es **positivo**")
elif numero < 0:
    print("El número es **negativo**")
else:
    print("El número es **cero**")
```

2. Determinar si un número es par o impar

```
numero = int(input("Ingresa un número: "))
if numero % 2 == 0:
    print("El número es **par**")
else:
    print("El número es **impar**")
```

3. Verificar si una persona es mayor de edad

```
edad = int(input("Ingresa tu edad: "))
if edad >= 18:
    print("Eres **mayor de edad**")
else:
    print("Eres **menor de edad**")
```

4. Comparar dos números e indicar cuál es mayor

```
a = float(input("Ingresa el primer número: "))
b = float(input("Ingresa el segundo número: "))
if a > b:
    print("El **primer número** es mayor")
elif b > a:
    print("El **segundo número** es mayor")
else:
    print("Ambos números son **iguales**")
```

5. Verificar si un año es bisiesto

```
anio = int(input("Ingresa un año: "))
if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):
    print("El año es **bisiesto**")
else:
    print("El año **no es bisiesto**")
```

6. Calcular el descuento según el monto de compra

```
monto = float(input("Ingresa el monto de compra: "))
if monto > 1000:
    descuento = monto * 0.10
    print(f"Descuento de **10%**: {descuento}")
else:
    print("No aplica **descuento**")
```

7. Determinar el estado de la temperatura

```
temp = float(input("Ingresa la temperatura en °C: "))
if temp < 0:
    print("Hace **frío extremo**")
elif temp <= 20:
    print("Hace **frío**")
elif temp <= 30:
    print("Hace **calor moderado**")
else:
    print("Hace **mucho calor**")
```

8. Verificar si un número está en un rango específico

```
numero = int(input("Ingresa un número entre 1 y 10: "))
if 1 <= numero <= 10:
    print("El número está dentro del **rango permitido**")
else:
    print("El número está **fuera del rango**")
```

9. Determinar el mayor de tres números

```
a = float(input("Número 1: "))
b = float(input("Número 2: "))
c = float(input("Número 3: "))
if a > b and a > c:
    print("El **primer número** es el mayor")
elif b > a and b > c:
    print("El **segundo número** es el mayor")
else:
    print("El **tercer número** es el mayor")
```

10. Indicar si un carácter es vocal o consonante

```
letra = input("Ingresa una letra: ").lower()
if letra in "aeiou":
    print("Es una **vocal**")
else:
    print("Es una **consonante**")
```

11. Determinar si una persona puede votar

```
edad = int(input("Ingresa tu edad: "))
if edad >= 18:
    print("Puedes **votar**")
else:
    print("No puedes **votar** todavía")
```

12. Revisar si un número es múltiplo de 5

```
numero = int(input("Ingresa un número: "))
if numero % 5 == 0:
    print("El número es **múltiplo de 5**")
else:
    print("El número **no es múltiplo de 5**")
```

13. Calcular tarifa de estacionamiento según horas

```
horas = int(input("Horas de estacionamiento: "))
if horas <= 2:
    tarifa = horas * 5
else:
    tarifa = 2 * 5 + (horas - 2) * 3
print(f"La tarifa es: **{tarifa}**")
```

14. Verificar si un número es positivo y par

```
numero = int(input("Ingresa un número: "))
if numero > 0 and numero % 2 == 0:
    print("El número es **positivo y par**")
else:
    print("El número **no cumple ambas condiciones**")
```

15. Clasificar edad en infantil, adolescente, adulto

```
edad = int(input("Ingresa tu edad: "))
if edad < 12:
    print("Eres **infantil**")
elif edad <= 18:
    print("Eres **adolescente**")
else:
    print("Eres **adulto**")
```

16. Determinar si un número es divisible entre 2 y 3

```
numero = int(input("Ingresa un número: "))
if numero % 2 == 0 and numero % 3 == 0:
    print("El número es divisible entre **2 y 3**")
else:
    print("El número **no es divisible entre 2 y 3**")
```

17. Indicar si un carácter es letra, número o símbolo

```
caracter = input("Ingresa un carácter: ")
if caracter.isalpha():
    print("Es una **letra**")
elif caracter.isdigit():
    print("Es un **número**")
else:
    print("Es un **símbolo**")
```

18. Calcular si hay recargo por edad en ticket de cine

```
edad = int(input("Ingresa tu edad: "))
precio = 100
if edad < 12 or edad >= 60:
    precio *= 0.8
    print("Precio con **descuento**:", precio)
else:
    print("Precio normal:", precio)
```

19. Verificar si un número está fuera de un rango

```
numero = int(input("Ingresa un número: "))
if numero < 10 or numero > 50:
    print("El número está **fuera del rango**")
else:
    print("El número está dentro del **rango**")
```

20. Indicar si un carácter es mayúscula o minúscula

```
caracter = input("Ingresa un carácter: ")
if caracter.isupper():
    print("El carácter es **mayúscula**")
else:
    print("El carácter es **minúscula**")
```

21. Determinar tipo de triángulo según lados

```
a = float(input("Lado 1: "))
b = float(input("Lado 2: "))
c = float(input("Lado 3: "))
if a == b == c:
    print("Es un triángulo **equilátero**")
elif a == b or b == c or a == c:
    print("Es un triángulo **isósceles**")
else:
    print("Es un triángulo **escaleno**")
```

22. Evaluar calificación de un estudiante

```
nota = float(input("Ingresa la nota (0-100): "))
if nota >= 90:
    print("Calificación: **Excelente**")
elif nota >= 75:
    print("Calificación: **Bueno**")
elif nota >= 60:
    print("Calificación: **Regular**")
else:
    print("Calificación: **Deficiente**")
```

23. Determinar si un número es divisible entre 4 o 6

```
numero = int(input("Ingresa un número: "))
if numero % 4 == 0 or numero % 6 == 0:
    print("El número es divisible entre **4 o 6**")
else:
    print("El número **no es divisible entre 4 o 6**")
```

24. Verificar si una cadena es igual a “Python”

```
texto = input("Escribe algo: ")
if texto.lower() == "python":
    print("¡Has escrito **Python**!")
else:
    print("No has escrito Python")
```

25. Determinar el signo zodiacal simplificado

```
mes = int(input("Mes de nacimiento (1-12): "))
dia = int(input("Día de nacimiento: "))
if (mes == 3 and dia >= 21) or (mes == 4 and dia <= 20):
    print("Signo: **Aries**")
else:
    print("Signo: Otro")
```

26. Revisar si el número ingresado es múltiplo de 7

```
numero = int(input("Ingresa un número: "))
if numero % 7 == 0:
    print("El número es **múltiplo de 7**")
else:
    print("El número **no es múltiplo de 7**")
```

27. Verificar contraseña básica

```
contraseña = input("Ingresa la contraseña: ")
if contraseña == "admin123":
    print("Acceso **permitido**")
else:
    print("Acceso **denegado**")
```

28. Indicar si el día ingresado es fin de semana

```
dia = input("Ingresa un día de la semana: ").lower()
if dia in ["sábado", "domingo"]:
    print("Es **fin de semana**")
else:
    print("Es un **día laboral**")
```

29. Evaluar si un número es múltiplo de 2, 3 o 5

```
numero = int(input("Ingresa un número: "))
if numero % 2 == 0:
    print("Múltiplo de 2")
if numero % 3 == 0:
    print("Múltiplo de 3")
if numero % 5 == 0:
    print("Múltiplo de 5")
```

30. Determinar tipo de triángulo según ángulos

```
a = int(input("Ángulo 1: "))
b = int(input("Ángulo 2: "))
c = int(input("Ángulo 3: "))
if a + b + c != 180:
    print("No es un triángulo válido")
elif a == b == c:
    print("Triángulo **equilátero**")
elif a == b or b == c or a == c:
    print("Triángulo **isósceles**")
else:
    print("Triángulo **escaleno**")
```

31. Determinar si un año es siglo

```
anio = int(input("Ingresa un año: "))
if anio % 100 == 0:
    print("Es un año de **siglo**")
else:
    print("No es un año de siglo")
```

32. Revisar si un número es positivo y mayor que 10

```
numero = int(input("Ingresa un número: "))
if numero > 0 and numero > 10:
    print("Número **positivo y mayor que 10**")
else:
    print("Número **no cumple la condición**")
```

33. Indicar si una letra es vocal, mayúscula y A

```
letra = input("Ingresa una letra: ")
if letra.lower() in "aeiou":
    print("Es **vocal**")
if letra.isupper():
    print("Es **mayúscula**")
if letra == "A":
    print("Es la letra **A**")
```

34. Determinar si se puede acceder a descuento especial

```
edad = int(input("Edad: "))
monto = float(input("Monto de compra: "))
if edad >= 65 or monto >= 500:
    print("Aplica **descuento especial**")
else:
    print("No aplica descuento")
```

35. Revisar si el número ingresado está en lista

```
numero = int(input("Ingresa un número: "))
lista = [2, 4, 6, 8, 10]
if numero in lista:
    print("Número **encontrado**")
else:
    print("Número **no encontrado**")
```

36. Calcular precio con IVA solo si es mayor a 100

```
precio = float(input("Precio: "))
if precio > 100:
    precio += precio * 0.21
    print("Precio con **IVA**:", precio)
else:
    print("Precio sin IVA:", precio)
```

37. Indicar si un número es múltiplo de 2 y 3 pero no de 5

```
numero = int(input("Número: "))
if numero % 2 == 0 and numero % 3 == 0 and numero % 5 != 0:
    print("Cumple la condición")
else:
    print("No cumple la condición")
```

38. Determinar si un carácter es letra mayúscula o minúscula

```
caracter = input("Ingresa un carácter: ")
if caracter.isalpha():
    if caracter.isupper():
        print("Letra **mayúscula**")
    else:
        print("Letra **minúscula**")
else:
    print("No es una letra")
```

39. Revisar si un número está en rango y es par

```
numero = int(input("Número: "))
if 10 <= numero <= 50 and numero % 2 == 0:
    print("Número **válido y par**")
else:
    print("Número **inválido o impar**")
```

40. Determinar si un carácter es un símbolo especial

```
caracter = input("Ingresa un carácter: ")
if not caracter.isalnum():
    print("Es un **símbolo especial**")
else:
    print("No es un símbolo especial")
```

---

3. LECCIÓN: ESTRUCTURAS REPETITIVAS (BUCLES) EN PYTHON

1. ¿Qué son las estructuras repetitivas?

Las **estructuras repetitivas**, también conocidas como **bucles**, permiten ejecutar un bloque de código **varias veces** sin necesidad de escribirlo repetidamente. Son fundamentales para automatizar tareas, procesar datos y resolver problemas de forma eficiente.

En Python existen principalmente dos tipos de bucles:

- **for**
- **while**

---

2. El bucle for

El bucle **for** se utiliza para **recorrer una secuencia de datos** como listas, cadenas, tuplas o rangos.

Sintaxis:
for variable in secuencia:
# instrucciones

Ejemplo:
for i in range(5):
print(i)

Resultado: imprime los números del **0 al 4**

Importante: la función **range()** genera una secuencia de números.

---

3. Uso de la función range()

La función **range()** es muy utilizada con el bucle for.

Formas de uso:

- range(fin) → desde 0 hasta fin-1
- range(inicio, fin) → desde inicio hasta fin-1
- range(inicio, fin, paso) → con incremento o decremento

Ejemplo:
for i in range(1, 11):
print(i) # imprime del 1 al 10

Ejemplo con paso:
for i in range(0, 10, 2):
print(i) # imprime 0, 2, 4, 6, 8

---

4. Recorrer listas con for

El bucle for permite recorrer elementos de una lista fácilmente.

Ejemplo:
numeros = [10, 20, 30, 40]

```
for num in numeros:
print(num)
```

---

5. Recorrer cadenas con for
También se pueden recorrer caracteres de una cadena.

Ejemplo:
```
texto = "Python"
for letra in texto:
print(letra)
```

---

6. El bucle while
El bucle **while** ejecuta un bloque de código **mientras una condición sea verdadera**.

Sintaxis:
```
while condición:
# instrucciones
```

Ejemplo:
```
i = 1
while i <= 5:
print(i)
i += 1
```

Importante: si la condición nunca se vuelve falsa, se genera un **bucle infinito**.

---

7. Bucle infinito
Un bucle infinito ocurre cuando la condición siempre es verdadera.

Ejemplo:
```
while True:
print("Esto se repite para siempre")
```

Para detenerlo manualmente se usa **Ctrl + C** o se debe incluir una condición de salida.

---

8. Uso de break
La instrucción **break** permite **salir inmediatamente del bucle**.

Ejemplo:
```
while True:
numero = int(input("Ingresa un número (0 para salir): "))
if numero == 0:
break
print("Número ingresado:", numero)
```

---

9. Uso de continue
La instrucción **continue** permite **saltar a la siguiente iteración del bucle**.

Ejemplo:
for i in range(1, 6):
if i == 3:
continue
print(i)

Resultado: imprime todos los números excepto el **3**

---

10. Uso de else en bucles
Los bucles pueden tener un bloque **else**, que se ejecuta cuando el bucle termina normalmente (sin break).

Ejemplo:
for i in range(5):
print(i)
else:
print("Bucle terminado correctamente")

---

11. Bucles anidados
Un **bucle anidado** es un bucle dentro de otro bucle.

Ejemplo:
for i in range(1, 4):
for j in range(1, 4):
print(i, j)

Esto genera combinaciones de valores.

---

12. Ejemplos prácticos con bucles

1. Imprimir números del 1 al 10
   for i in range(1, 11):
   print(i)
2. Sumar números del 1 al N
   N = int(input("Ingresa N: "))
   suma = 0
   for i in range(1, N+1):
   suma += i
   print("Suma:", suma)
3. Tabla de multiplicar
   numero = int(input("Número: "))
   for i in range(1, 11):
   print(numero, "x", i, "=", numero*i)

4. Contar vocales en una cadena
   texto = input("Texto: ")
   contador = 0
   for letra in texto:
   if letra.lower() in "aeiou":
   contador += 1
   print("Vocales:", contador)
5. Factorial de un número
   n = int(input("Número: "))
   factorial = 1
   for i in range(1, n+1):
   factorial *= i
   print("Factorial:", factorial)

---

13. Control de iteraciones
En los bucles es importante controlar:

- El **inicio**
- La **condición de parada**
- El **incremento o decremento**

Ejemplo:
i = 1
while i <= 10:
print(i)
i += 1

---

14. Buenas prácticas con bucles

- Evitar bucles infinitos sin control
- Usar nombres de variables claros (contador, i, total)
- Usar break solo cuando sea necesario
- Mantener el código legible
- Evitar anidar demasiados bucles (puede complicar el código)

---

15. Diferencias entre for y while

- **for**: se usa cuando se conoce el número de repeticiones
- **while**: se usa cuando depende de una condición

Ejemplo:
for i in range(5): → número fijo de repeticiones
while numero != 0: → depende de una condición

---

16. Aplicaciones prácticas de los bucles
Los bucles se utilizan para:

- Procesar listas de datos
- Validar entradas del usuario
- Realizar cálculos repetitivos
- Generar patrones (figuras, tablas)
- Automatizar tareas

---

17. Ejercicios sugeridos para practicar

1. Imprimir los números pares del 1 al 100
2. Calcular la suma de los números impares
3. Pedir números hasta que el usuario ingrese 0
4. Crear un programa que muestre los divisores de un número
5. Generar una pirámide de asteriscos
6. Contar cuántos números positivos y negativos se ingresan

---

18. Resumen de estructuras repetitivas

- **for**: recorre secuencias
- **while**: repite mientras la condición sea verdadera
- **break**: termina el bucle
- **continue**: salta a la siguiente iteración
- **else en bucles**: se ejecuta al finalizar correctamente
- **Bucles anidados**: permiten estructuras más complejas

Esta lección incluye:

- **Concepto de bucles**
- **Uso de for y while**
- **Control de flujo con break y continue**
- **Bucles anidados**
- **Ejemplos prácticos paso a paso**
- **Buenas prácticas y aplicaciones reales**

---

3. ESTRUCTURAS REPETITIVAS. BUCLES

1. Imprimir números del 1 al 10

```
for i in range(1, 11):
    print(i)
```

2. Sumar los primeros N números

```
N = int(input("Ingresa N: "))
suma = 0
for i in range(1, N+1):
    suma += i
print("La suma es:", suma)
```

3. Imprimir números pares del 1 al 20

```
for i in range(2, 21, 2):
    print(i)
```

4. Contar cuántos números negativos hay en una lista

```
numeros = [3, -2, 5, -7, 0]
contador = 0
for num in numeros:
    if num < 0:
        contador += 1
print("Números negativos:", contador)
```

5. Imprimir tabla de multiplicar de un número

```
numero = int(input("Ingresa un número: "))
for i in range(1, 11):
    print(f"{numero} x {i} = {numero*i}")
```

6. Sumar los números pares entre 1 y 50

```
suma = 0
for i in range(2, 51, 2):
    suma += i
print("Suma de números pares:", suma)
```

7. Contar cuántas veces aparece una letra en una cadena

```
texto = input("Ingresa un texto: ")
letra = input("Ingresa la letra a buscar: ")
contador = 0
for c in texto:
    if c == letra:
        contador += 1
print(f"La letra {letra} aparece {contador} veces")
```

8. Mostrar los primeros N términos de la serie de Fibonacci

```
N = int(input("Número de términos: "))
a, b = 0, 1
for _ in range(N):
    print(a)
    a, b = b, a + b
```

9. Imprimir números del 10 al 1 (regresivo)

```
for i in range(10, 0, -1):
    print(i)
```

10. Sumar solo los números positivos de una lista

```
numeros = [-2, 5, 7, -1, 3]
suma = 0
for num in numeros:
    if num > 0:
        suma += num
print("Suma de positivos:", suma)
```

11. Pedir 5 números e indicar cuál es mayor

```
mayor = None
for i in range(5):
    num = float(input(f"Número {i+1}: "))
    if mayor is None or num > mayor:
        mayor = num
print("El mayor es:", mayor)
```

12. Pedir 5 números e indicar cuál es menor

```
menor = None
for i in range(5):
    num = float(input(f"Número {i+1}: "))
    if menor is None or num < menor:
        menor = num
print("El menor es:", menor)
```

13. Calcular factorial de un número

```
numero = int(input("Ingresa un número: "))
factorial = 1
for i in range(1, numero+1):
    factorial *= i
print(f"Factorial de {numero} es {factorial}")
```

14. Mostrar todos los divisores de un número

```
numero = int(input("Ingresa un número: "))
print("Divisores:")
for i in range(1, numero+1):
    if numero % i == 0:
        print(i)
```

15. Imprimir números impares entre 1 y 20

```
for i in range(1, 21, 2):
    print(i)
```

16. Pedir N números y calcular la media

```
N = int(input("Cuántos números: "))
suma = 0
for i in range(N):
    num = float(input(f"Número {i+1}: "))
    suma += num
media = suma / N
print("Media:", media)
```

17. Contar cuántos números pares y cuántos impares hay en una lista

```
numeros = [1,2,3,4,5,6]
pares = impares = 0
for num in numeros:
    if num % 2 == 0:
        pares += 1
    else:
        impares += 1
print("Pares:", pares, "Impares:", impares)
```

18. Imprimir un triángulo de asteriscos

```
n = int(input("Número de filas: "))
for i in range(1, n+1):
    print("*" * i)
```

19. Imprimir un triángulo invertido de asteriscos

```
n = int(input("Número de filas: "))
for i in range(n, 0, -1):
    print("*" * i)
```

20. Sumar los números de una lista hasta que se ingrese cero

```
suma = 0
while True:
    num = int(input("Ingresa un número (0 para salir): "))
    if num == 0:
        break
    suma += num
print("Suma total:", suma)
```

21. Mostrar los números del 1 al N usando while

```
N = int(input("Ingresa N: "))
i = 1
while i <= N:
    print(i)
    i += 1
```

22. Pedir números hasta que se ingrese negativo y mostrar el promedio

```
suma = 0
contador = 0
while True:
    num = float(input("Ingresa un número (negativo para salir): "))
    if num < 0:
        break
    suma += num
    contador += 1
if contador > 0:
    print("Promedio:", suma/contador)
else:
    print("No se ingresaron números válidos")
```

23. Contar cuántas vocales hay en una cadena

```
texto = input("Ingresa un texto: ")
vocales = "aeiouAEIOU"
contador = 0
for c in texto:
```

```
    if c in vocales:
        contador += 1
print("Número de vocales:", contador)
```

24. Pedir N números y mostrar los que son positivos

```
N = int(input("Cuántos números: "))
for i in range(N):
    num = float(input(f"Número {i+1}: "))
    if num > 0:
        print("Positivo:", num)
```

25. Mostrar los primeros 10 números cuadrados

```
for i in range(1, 11):
    print(f"{i}^2 = {i**2}")
```

26. Sumar los números del 1 al 100 que sean múltiplos de 3

```
suma = 0
for i in range(1, 101):
    if i % 3 == 0:
        suma += i
print("Suma de múltiplos de 3:", suma)
```

27. Pedir N números e indicar cuántos son mayores que 10

```
N = int(input("Cuántos números: "))
contador = 0
for i in range(N):
    num = float(input(f"Número {i+1}: "))
    if num > 10:
        contador += 1
print("Números mayores a 10:", contador)
```

28. Imprimir los primeros N términos de la serie 2, 4, 6, 8…

```
N = int(input("Número de términos: "))
for i in range(1, N+1):
    print(2*i)
```

29. Pedir contraseñas hasta acertar

```
while True:
    contraseña = input("Ingresa contraseña: ")
    if contraseña == "python123":
        print("¡Acceso concedido!")
        break
    else:
        print("Contraseña incorrecta")
```

30. Contar cuántos números impares hay entre 1 y N

```
N = int(input("Ingresa N: "))
contador = 0
for i in range(1, N+1):
    if i % 2 != 0:
        contador += 1
print("Números impares:", contador)
```

31. Mostrar números del 1 al 50 saltando de 5 en 5

```
for i in range(1, 51, 5):
    print(i)
```

32. Pedir N números y mostrar solo los negativos

```
N = int(input("Cuántos números: "))
for i in range(N):
    num = float(input(f"Número {i+1}: "))
    if num < 0:
        print("Negativo:", num)
```

33. Imprimir una pirámide de números

```
n = int(input("Número de filas: "))
for i in range(1, n+1):
    for j in range(1, i+1):
        print(j, end=" ")
    print()
```

34. Calcular la suma de los cuadrados hasta N

```
N = int(input("Ingresa N: "))
suma = 0
for i in range(1, N+1):
    suma += i**2
print("Suma de cuadrados:", suma)
```

35. Pedir números hasta que la suma supere 100

```
suma = 0
while suma <= 100:
    num = int(input("Ingresa un número: "))
    suma += num
print("Suma total:", suma)
```

36. Imprimir los primeros N múltiplos de 7

```
N = int(input("Número de múltiplos: "))
for i in range(1, N+1):
    print(7*i)
```

37. Mostrar números primos hasta N

```
N = int(input("Ingresa N: "))
for num in range(2, N+1):
    primo = True
    for i in range(2, int(num**0.5)+1):
        if num % i == 0:
            primo = False
            break
    if primo:
        print(num)
```

38. Contar cuántas veces aparece un número en una lista

```
lista = [1,2,2,3,4,2,5]
num = int(input("Número a buscar: "))
contador = 0
for n in lista:
    if n == num:
        contador += 1
print("Aparece", contador, "veces")
```

39. Pedir 5 números e indicar cuántos son pares y cuántos impares

```
pares = impares = 0
for i in range(5):
    num = int(input(f"Número {i+1}: "))
    if num % 2 == 0:
        pares += 1
    else:
        impares += 1
print("Pares:", pares, "Impares:", impares)
```

40. Imprimir un patrón de asteriscos en forma de cuadrado

```
n = int(input("Tamaño del cuadrado: "))
for i in range(n):
    print("* " * n)
```

---

4. LECCIÓN: ESTRUCTURAS DE DATOS EN PYTHON – LISTAS (LISTS)

1. ¿Qué es una lista en Python?

Una **lista** es una estructura de datos que permite **almacenar múltiples elementos en una sola variable**. Estos elementos pueden ser de **diferentes tipos de datos**, como números, cadenas, booleanos, e incluso otras listas.

Ejemplo:
lista = [1, 2, 3, 4]
nombres = ["Ana", "Luis", "Pedro"]
mixta = [1, "Hola", True, 3.5]

Importante: las listas son **ordenadas, modificables (mutables)** y permiten elementos duplicados.

---

2. Crear listas

Las listas se crean usando **corchetes []**.

Ejemplo:
numeros = [10, 20, 30]
frutas = ["manzana", "pera", "naranja"]

También se puede usar el constructor:
lista = list((1, 2, 3))

---

3. Acceder a elementos de una lista

Cada elemento tiene un **índice**, que comienza en **0**.

Ejemplo:
frutas = ["manzana", "pera", "naranja"]
print(frutas[0]) # manzana
print(frutas[1]) # pera

También se pueden usar índices negativos:
print(frutas[-1]) # naranja

---

4. Modificar elementos de una lista

Las listas son **mutables**, por lo que se pueden cambiar sus elementos.

Ejemplo:
frutas = ["manzana", "pera", "naranja"]
frutas[1] = "uva"
print(frutas) # ["manzana", "uva", "naranja"]

---

5. Recorrer listas

Se pueden recorrer usando bucles.

Con for:
for fruta in frutas:
print(fruta)

Con índice:
for i in range(len(frutas)):
print(frutas[i])

---

6. Agregar elementos a una lista

Métodos principales:

- **append()** → agrega un elemento al final
- **insert()** → agrega en una posición específica
- **extend()** → agrega varios elementos

Ejemplo:
numeros = [1, 2, 3]
numeros.append(4)
numeros.insert(1, 10)
numeros.extend([5, 6])
print(numeros)

---

7. Eliminar elementos de una lista

Métodos:

- **remove()** → elimina un valor específico
- **pop()** → elimina por índice
- **del** → elimina elementos o la lista completa
- **clear()** → vacía la lista

Ejemplo:
numeros = [1, 2, 3, 4]
numeros.remove(2)
numeros.pop(1)
del numeros[0]
numeros.clear()

---

8. Longitud de una lista
Se usa la función **len()** para saber cuántos elementos tiene.

Ejemplo:
lista = [1, 2, 3, 4]
print(len(lista)) # 4

---

9. Buscar elementos en una lista

Se puede usar:

- **in** → verifica si existe
- **index()** → obtiene la posición

Ejemplo:
lista = [10, 20, 30]
print(20 in lista) # True
print(lista.index(30)) # 2

---

10. Contar elementos repetidos
Se usa el método **count()**.

Ejemplo:
lista = [1, 2, 2, 3, 2]
print(lista.count(2)) # 3

---

11. Ordenar listas

Métodos:

- **sort()** → ordena la lista original
- **sorted()** → devuelve una nueva lista ordenada

Ejemplo:
numeros = [5, 2, 8, 1]
numeros.sort()
print(numeros)

Orden descendente:
numeros.sort(reverse=True)

---

12. Invertir listas
Se usa el método **reverse()**.

Ejemplo:
lista = [1, 2, 3]
lista.reverse()
print(lista) # [3, 2, 1]

---

13. Copiar listas
Para evitar modificar la original, se puede copiar.

Ejemplo:
lista1 = [1, 2, 3]
lista2 = lista1.copy()

---

14. Listas anidadas
Una lista puede contener otras listas.

Ejemplo:
matriz = [[1, 2], [3, 4], [5, 6]]
print(matriz[0][1]) # 2

---

15. Slicing (rebanado de listas)
Permite obtener partes de una lista.

Sintaxis:
lista[inicio:fin:paso]

Ejemplo:
numeros = [1, 2, 3, 4, 5]
print(numeros[1:4]) # [2, 3, 4]
print(numeros[:3]) # [1, 2, 3]
print(numeros[::2]) # [1, 3, 5]

---

16. List comprehension (comprensión de listas)
Es una forma **compacta y eficiente** de crear listas.

Ejemplo:
cuadrados = [x**2 for x in range(5)]
print(cuadrados) # [0, 1, 4, 9, 16]

Con condición:
pares = [x for x in range(10) if x % 2 == 0]

---

17. Operaciones comunes con listas

Concatenación:
lista1 = [1, 2]
lista2 = [3, 4]
print(lista1 + lista2)

Repetición:
print([1, 2] * 3) # [1, 2, 1, 2, 1, 2]

---

18. Buenas prácticas con listas

- Usar nombres descriptivos

- Evitar modificar listas mientras se recorren
- Usar list comprehension cuando sea posible
- No duplicar datos innecesariamente

---

19. Ejemplos prácticos con listas

1. Sumar todos los elementos de una lista
   numeros = [1, 2, 3, 4]
   suma = 0
   for n in numeros:
   suma += n
   print("Suma:", suma)
2. Encontrar el mayor número
   numeros = [5, 9, 2, 8]
   mayor = numeros[0]
   for n in numeros:
   if n > mayor:
   mayor = n
   print("Mayor:", mayor)
3. Contar números pares
   numeros = [1, 2, 3, 4, 5, 6]
   contador = 0
   for n in numeros:
   if n % 2 == 0:
   contador += 1
   print("Pares:", contador)
4. Eliminar duplicados
   lista = [1, 2, 2, 3, 4, 4]
   sin_duplicados = list(set(lista))
   print(sin_duplicados)
5. Invertir una lista manualmente
   lista = [1, 2, 3]
   invertida = []
   for i in range(len(lista)-1, -1, -1):
   invertida.append(lista[i])
   print(invertida)

---

20. Resumen de listas en Python

- Las listas permiten almacenar **múltiples datos**
- Son **mutables y ordenadas**
- Permiten acceso por índice

- Se pueden modificar, agregar y eliminar elementos
- Incluyen muchos métodos útiles como append, remove, sort
- Son fundamentales para trabajar con datos en Python

Esta lección incluye:

- **Concepto de listas**
- **Creación, acceso y modificación**
- **Métodos principales**
- **Recorridos con bucles**
- **Listas anidadas y slicing**
- **List comprehension**
- **Ejemplos prácticos reales**

---

1. Crear una lista e imprimir sus elementos

```
lista = [1, 2, 3, 4, 5]
for elemento in lista:
    print(elemento)
```

2. Calcular la suma de los elementos de una lista

```
lista = [1, 2, 3, 4, 5]
suma = 0
for n in lista:
    suma += n
print("Suma:", suma)
```

3. Calcular el promedio de una lista

```
lista = [10, 20, 30]
promedio = sum(lista) / len(lista)
print("Promedio:", promedio)
```

4. Encontrar el número mayor en una lista

```
lista = [5, 9, 2, 8]
mayor = lista[0]
for n in lista:
    if n > mayor:
        mayor = n
print("Mayor:", mayor)
```

5. Encontrar el número menor en una lista

```
lista = [5, 9, 2, 8]
menor = lista[0]
for n in lista:
    if n < menor:
        menor = n
print("Menor:", menor)
```

6. Contar elementos en una lista

```
lista = [1, 2, 3, 4]
print("Cantidad:", len(lista))
```

7. Verificar si un elemento está en la lista

```
lista = [1, 2, 3]
numero = 2
if numero in lista:
    print("Está en la lista")
```

8. Agregar un elemento a la lista

```
lista = [1, 2, 3]
lista.append(4)
print(lista)
```

9. Eliminar un elemento de la lista

```
lista = [1, 2, 3]
lista.remove(2)
print(lista)
```

10. Invertir una lista

```
lista = [1, 2, 3]
lista.reverse()
print(lista)
```

---

11. Ordenar una lista

```
lista = [5, 2, 8, 1]
lista.sort()
print(lista)
```

12. Contar números pares en una lista

```
lista = [1, 2, 3, 4, 5]
contador = 0
for n in lista:
    if n % 2 == 0:
        contador += 1
print("Pares:", contador)
```

13. Crear lista con números del 1 al 10

```
lista = list(range(1, 11))
print(lista)
```

14. Multiplicar todos los elementos de una lista

```
lista = [1, 2, 3, 4]
resultado = 1
for n in lista:
    resultado *= n
print("Multiplicación:", resultado)
```

15. Copiar una lista

```
lista1 = [1, 2, 3]
lista2 = lista1.copy()
print(lista2)
```

16. Concatenar dos listas

```
a = [1, 2]
b = [3, 4]
c = a + b
print(c)
```

17. Encontrar índice de un elemento

```
lista = [10, 20, 30]
print(lista.index(20))
```

18. Contar repeticiones de un elemento

```
lista = [1, 2, 2, 3]
print(lista.count(2))
```

19. Eliminar duplicados

```
lista = [1, 2, 2, 3]
sin_dup = list(set(lista))
print(sin_dup)
```

20. Crear lista de cuadrados

```
cuadrados = [x**2 for x in range(1, 6)]
print(cuadrados)
```

---

21. Filtrar números mayores a 5

```
lista = [1, 6, 3, 8]
mayores = [x for x in lista if x > 5]
print(mayores)
```

22. Sumar solo números positivos

```
lista = [-1, 2, -3, 4]
suma = 0
for n in lista:
    if n > 0:
        suma += n
print(suma)
```

23. Reemplazar un valor en lista

```
lista = [1, 2, 3]
lista[1] = 10
print(lista)
```

24. Crear lista con input del usuario

```
lista = []
for i in range(3):
    lista.append(int(input("Número: ")))
print(lista)
```

25. Sumar dos listas elemento a elemento

```
a = [1, 2, 3]
b = [4, 5, 6]
resultado = []
for i in range(len(a)):
    resultado.append(a[i] + b[i])
print(resultado)
```

26. Encontrar números impares

```
lista = [1, 2, 3, 4]
impares = [x for x in lista if x % 2 != 0]
print(impares)
```

27. Ordenar lista descendente

```
lista = [5, 2, 8]
lista.sort(reverse=True)
print(lista)
```

28. Insertar elemento en posición específica

```
lista = [1, 2, 3]
lista.insert(1, 10)
print(lista)
```

29. Eliminar último elemento

```
lista = [1, 2, 3]
lista.pop()
print(lista)
```

30. Contar negativos

```
lista = [-1, 2, -3]
contador = sum(1 for x in lista if x < 0)
print(contador)
```

---

31. Crear lista de letras

```
lista = list("python")
print(lista)
```

32. Unir lista en cadena

```
lista = ["Hola", "Mundo"]
print(" ".join(lista))
```

33. Buscar máximo con función

```
lista = [3, 7, 2]
print(max(lista))
```

34. Buscar mínimo con función

```
lista = [3, 7, 2]
print(min(lista))
```

35. Multiplicar lista por número

```
lista = [1, 2, 3]
resultado = [x*2 for x in lista]
print(resultado)
```

36. Crear lista de números aleatorios

```
import random
lista = [random.randint(1,10) for _ in range(5)]
print(lista)
```

37. Filtrar números menores a 0

```
lista = [-5, 2, -1, 3]
negativos = [x for x in lista if x < 0]
print(negativos)
```

38. Sumar elementos en posiciones pares

```
lista = [1,2,3,4]
suma = sum(lista[i] for i in range(0,len(lista),2))
print(suma)
```

39. Repetir lista

```
lista = [1,2]
print(lista * 3)
```

40. Crear lista invertida manualmente

```
lista = [1,2,3]
invertida = []
for i in range(len(lista)-1,-1,-1):
    invertida.append(lista[i])
print(invertida)
```

Si quieres, continúo con los **programas 41 al 100**, donde subiré el nivel incluyendo:

- **listas anidadas (matrices)**
- **procesamiento de datos reales**
- **problemas tipo examen**

Solo dime: **“continúa”** 👍

5. LECCIÓN: ESTRUCTURAS DE DATOS EN PYTHON – CONJUNTOS (SETS)

1. ¿Qué es un conjunto en Python?

Un **conjunto (set)** es una estructura de datos que permite almacenar **elementos únicos**, es decir, **no admite duplicados**.

Los conjuntos son:

- **No ordenados**
- **Mutables (se pueden modificar)**
- **No permiten elementos repetidos**

Ejemplo:

```
conjunto = {1, 2, 3, 4}
print(conjunto)
```

Si se repiten valores:

```
conjunto = {1, 2, 2, 3}
print(conjunto) # Resultado: {1, 2, 3}
```

---

2. Crear conjuntos

Se pueden crear usando **llaves {}** o la función **set()**.

Ejemplo:

```
a = {1, 2, 3}
b = set([4, 5, 6])
```

Para crear un conjunto vacío:

```
c = set()
```

Importante: usar {} crea un diccionario vacío, no un conjunto.

---

3. Características importantes de los conjuntos

- No tienen **índices** (no se accede como listas)
- No mantienen el orden
- Son útiles para eliminar duplicados
- Permiten operaciones matemáticas como unión e intersección

---

4. Agregar elementos a un conjunto

Métodos:

- **add()** → agrega un solo elemento
- **update()** → agrega varios elementos

Ejemplo:
conjunto = {1, 2}
conjunto.add(3)
conjunto.update([4, 5])
print(conjunto)

---

5. Eliminar elementos de un conjunto

Métodos:

- **remove()** → elimina un elemento (error si no existe)
- **discard()** → elimina sin error
- **pop()** → elimina un elemento aleatorio
- **clear()** → elimina todos los elementos

Ejemplo:
conjunto = {1, 2, 3}
conjunto.remove(2)
conjunto.discard(5)
conjunto.pop()
conjunto.clear()

---

6. Recorrer un conjunto
Se puede recorrer con bucles.

Ejemplo:
conjunto = {1, 2, 3}
for elemento in conjunto:
print(elemento)

---

7. Verificar pertenencia en un conjunto
Se usa **in** o **not in**.

Ejemplo:
conjunto = {1, 2, 3}
print(2 in conjunto)
print(5 not in conjunto)

---

8. Operaciones entre conjuntos

Los conjuntos permiten realizar operaciones matemáticas:

Unión (union) → combina elementos sin repetir
Ejemplo:
a = {1, 2}

b = {2, 3}
print(a.union(b)) # {1, 2, 3}

También:
print(a | b)

Intersección (intersection) → elementos comunes
Ejemplo:
print(a.intersection(b)) # {2}
print(a & b)

Diferencia (difference) → elementos de un conjunto que no están en otro
Ejemplo:
print(a.difference(b)) # {1}
print(a - b)

Diferencia simétrica (symmetric_difference) → elementos que no coinciden
Ejemplo:
print(a.symmetric_difference(b)) # {1, 3}
print(a ^ b)

---

9. Subconjuntos y superconjuntos

Métodos:

- **issubset()** → verifica si un conjunto está dentro de otro
- **issuperset()** → verifica si contiene a otro

Ejemplo:
a = {1, 2}
b = {1, 2, 3}
print(a.issubset(b)) # True
print(b.issuperset(a)) # True

---

10. Copiar conjuntos
Se usa **copy()**

Ejemplo:
a = {1, 2, 3}
b = a.copy()

---

11. Conjuntos inmutables (frozenset)
Un **frozenset** es un conjunto que **no se puede modificar**.

Ejemplo:
a = frozenset([1, 2, 3])

a.add(4) → error

---

12. Aplicaciones prácticas de los conjuntos

1. Eliminar duplicados de una lista
   lista = [1, 2, 2, 3]
   sin_dup = list(set(lista))
   print(sin_dup)
2. Encontrar elementos comunes
   a = [1, 2, 3]
   b = [2, 3, 4]
   comunes = set(a) & set(b)
   print(comunes)
3. Diferencia entre listas
   a = [1, 2, 3]
   b = [2, 4]
   print(set(a) - set(b))

---

13. Ejemplos prácticos con conjuntos

1. Crear un conjunto desde entrada del usuario
   numeros = set()
   for i in range(5):
   numeros.add(int(input("Número: ")))
   print(numeros)
2. Contar elementos únicos
   lista = [1, 1, 2, 2, 3]
   print(len(set(lista)))
3. Verificar si dos listas tienen elementos en común
   a = [1, 2, 3]
   b = [4, 5, 3]
   if set(a) & set(b):
   print("Tienen elementos en común")
4. Unir varias listas sin duplicados
   a = [1, 2]
   b = [2, 3]
   c = list(set(a + b))
   print(c)

5. Verificar subconjunto
   a = {1, 2}
   b = {1, 2, 3}
   if a.issubset(b):
   print("Es subconjunto")

---

14. Buenas prácticas con conjuntos

- Usarlos cuando se necesiten **elementos únicos**
- Evitar usarlos cuando el orden sea importante
- Aprovechar operaciones como unión e intersección
- Usar conjuntos para **comparaciones rápidas**

---

15. Diferencias entre listas y conjuntos

- Lista: ordenada, permite duplicados
- Conjunto: no ordenado, no permite duplicados
- Lista: acceso por índice
- Conjunto: no tiene índice

---

16. Ventajas de los conjuntos

- Eliminan duplicados automáticamente
- Operaciones rápidas
- Ideales para análisis de datos
- Sintaxis sencilla

---

17. Resumen de conjuntos

- Los **conjuntos** almacenan elementos únicos
- No tienen orden ni índices
- Permiten operaciones matemáticas
- Son útiles para eliminar duplicados y comparar datos
- Incluyen métodos como add, remove, union, intersection

Esta lección incluye:

- **Concepto de conjuntos**

- **Creación y manipulación**
- **Operaciones matemáticas entre conjuntos**
- **Aplicaciones prácticas reales**
- **Comparación con listas**
- **Ejemplos completos paso a paso**

---

1. Crear un conjunto e imprimir sus elementos

```
conjunto = {1, 2, 3}
for elemento in conjunto:
    print(elemento)
```

2. Eliminar duplicados de una lista usando conjunto

```
lista = [1, 2, 2, 3, 3]
resultado = set(lista)
print(resultado)
```

3. Contar elementos únicos

```
lista = [1, 1, 2, 2, 3]
print(len(set(lista)))
```

4. Agregar elementos a un conjunto

```
conjunto = {1, 2}
conjunto.add(3)
print(conjunto)
```

5. Agregar varios elementos

```
conjunto = {1, 2}
conjunto.update([3, 4, 5])
print(conjunto)
```

6. Eliminar un elemento

```
conjunto = {1, 2, 3}
conjunto.remove(2)
print(conjunto)
```

7. Eliminar elemento sin error

```
conjunto = {1, 2, 3}
conjunto.discard(5)
print(conjunto)
```

8. Unión de conjuntos

```
a = {1, 2}
b = {2, 3}
print(a | b)
```

9. Intersección de conjuntos

```
a = {1, 2}
b = {2, 3}
print(a & b)
```

10. Diferencia de conjuntos

```
a = {1, 2, 3}
b = {2}
print(a - b)
```

---

11. Diferencia simétrica

```
a = {1, 2}
b = {2, 3}
print(a ^ b)
```

12. Verificar pertenencia

```
conjunto = {1, 2, 3}
print(2 in conjunto)
```

13. Recorrer un conjunto

```
conjunto = {10, 20, 30}
for x in conjunto:
    print(x)
```

14. Crear conjunto desde input

```
conjunto = set()
for i in range(3):
    conjunto.add(int(input("Número: ")))
print(conjunto)
```

15. Vaciar un conjunto

```
conjunto = {1, 2, 3}
conjunto.clear()
print(conjunto)
```

16. Copiar un conjunto

```
a = {1, 2}
b = a.copy()
print(b)
```

17. Encontrar elementos comunes entre listas

```
a = [1, 2, 3]
b = [2, 4]
print(set(a) & set(b))
```

18. Unir listas sin duplicados

```
a = [1, 2]
b = [2, 3]
print(set(a + b))
```

19. Verificar subconjunto

```
a = {1, 2}
b = {1, 2, 3}
print(a.issubset(b))
```

20. Verificar superconjunto

```
a = {1, 2, 3}
b = {1, 2}
print(a.issuperset(b))
```

---

21. Convertir texto a conjunto de letras únicas

```
texto = "python"
print(set(texto))
```

22. Contar vocales únicas en un texto

```
texto = input("Texto: ")
vocales = set("aeiou")
print(len(set(texto.lower()) & vocales))
```

23. Eliminar elementos comunes entre dos listas

```
a = [1, 2, 3]
b = [2, 3]
print(set(a) - set(b))
```

24. Encontrar elementos diferentes

```
a = {1, 2}
b = {2, 3}
print(a ^ b)
```

25. Verificar si dos listas son iguales (sin importar orden)

```
a = [1, 2, 3]
b = [3, 2, 1]
print(set(a) == set(b))
```

26. Crear conjunto de números pares

```
pares = {x for x in range(10) if x % 2 == 0}
print(pares)
```

27. Crear conjunto de cuadrados

```
cuadrados = {x**2 for x in range(5)}
print(cuadrados)
```

28. Eliminar duplicados y ordenar

```
lista = [3, 1, 2, 2]
resultado = sorted(set(lista))
print(resultado)
```

29. Contar letras únicas en una frase

```
texto = input("Texto: ")
print(len(set(texto)))
```

30. Detectar caracteres repetidos

```
texto = "programacion"
repetidos = {x for x in texto if texto.count(x) > 1}
print(repetidos)
```

---

31. Verificar si dos conjuntos son disjuntos

```
a = {1, 2}
b = {3, 4}
print(a.isdisjoint(b))
```

32. Obtener elementos únicos de varias listas

```
a = [1, 2]
b = [2, 3]
c = [3, 4]
print(set(a) | set(b) | set(c))
```

33. Contar números únicos ingresados

```
datos = [int(input("Número: ")) for _ in range(5)]
print(len(set(datos)))
```

34. Filtrar elementos mayores a 5

```
conjunto = {1, 6, 3, 8}
print({x for x in conjunto if x > 5})
```

35. Crear conjunto de números negativos

```
lista = [-1, 2, -3, 4]
print({x for x in lista if x < 0})
```

36. Encontrar números en común entre tres listas

```
a = [1, 2, 3]
b = [2, 3, 4]
c = [3, 4, 5]
print(set(a) & set(b) & set(c))
```

37. Generar conjunto de múltiplos de 3

```
multiplos = {x for x in range(1, 20) if x % 3 == 0}
print(multiplos)
```

38. Verificar si un elemento no está en el conjunto

```
conjunto = {1, 2, 3}
print(5 not in conjunto)
```

39. Eliminar un elemento aleatorio

```
conjunto = {1, 2, 3}
conjunto.pop()
print(conjunto)
```

40. Convertir lista a conjunto y volver a lista

```
lista = [1, 2, 2, 3]
lista = list(set(lista))
print(lista)
```

---

41. Crear conjunto de letras únicas sin espacios

```
texto = "hola mundo"
print(set(texto.replace(" ", "")))
```

42. Obtener diferencia entre múltiples conjuntos

```
a = {1, 2, 3}
b = {2}
c = {3}
print(a - b - c)
```

43. Comparar dos conjuntos

```
a = {1, 2}
b = {1, 2}
print(a == b)
```

44. Crear conjunto de números ingresados hasta 0

```
conjunto = set()
while True:
    n = int(input("Número: "))
    if n == 0:
        break
    conjunto.add(n)
print(conjunto)
```

45. Filtrar palabras únicas

```
texto = "hola hola mundo"
print(set(texto.split()))
```

46. Encontrar letras comunes entre dos palabras

```
a = input("Palabra 1: ")
b = input("Palabra 2: ")
print(set(a) & set(b))
```

47. Obtener letras diferentes entre dos palabras

```
print(set(a) ^ set(b))
```

48. Generar conjunto de números del 1 al 100

```
print(set(range(1, 101)))
```

49. Contar elementos únicos mayores a 10

```
lista = [5, 10, 15, 20]
print(len({x for x in lista if x > 10}))
```

50. Crear conjunto con números al cuadrado únicos

```
lista = [1, 2, 2, 3]
print({x**2 for x in lista})
```

---

6. LECCIÓN: ESTRUCTURAS DE DATOS EN PYTHON – TUPLAS (TUPLES)

1. ¿QUÉ ES UNA TUPLA EN PYTHON?

Una **tupla (tuple)** es una estructura de datos que permite almacenar varios elementos en una sola variable.

Las tuplas son:

- **Ordenadas**
- **Inmutables (no se pueden modificar después de crearse)**
- **Permiten elementos duplicados**

Ejemplo:
tupla = (1, 2, 3)
print(tupla)

---

2. CARACTERÍSTICAS IMPORTANTES DE LAS TUPLAS

- Mantienen el **orden de los elementos**
- Son **inmutables**, no se pueden cambiar
- Permiten diferentes tipos de datos
- Son más **rápidas que las listas**

Ejemplo:
tupla = (1, "hola", 3.5, True)

---

3. CREACIÓN DE TUPLAS

Se crean usando **paréntesis ()**

Ejemplo:
a = (1, 2, 3)

También se pueden crear sin paréntesis:
b = 4, 5, 6

---

4. TUPLA CON UN SOLO ELEMENTO

Para crear una tupla de un solo elemento se necesita una **coma**

Ejemplo:
tupla = (5,)

Importante: sin la coma no es una tupla

---

5. ACCESO A LOS ELEMENTOS

Se accede mediante índices

Ejemplo:
tupla = (10, 20, 30)
print(tupla[0])
print(tupla[1])

---

6. ÍNDICES NEGATIVOS
Permiten acceder desde el final

Ejemplo:
tupla = (10, 20, 30)
print(tupla[-1])

---

7. REBANADO (SLICING)
Permite obtener partes de la tupla

Ejemplo:
tupla = (1, 2, 3, 4, 5)
print(tupla[1:4])

---

8. RECORRER UNA TUPLA
Se puede usar un bucle

Ejemplo:
tupla = (1, 2, 3)
for elemento in tupla:
print(elemento)

---

9. FUNCIONES BÁSICAS CON TUPLAS

- **len()** → longitud
- **max()** → valor máximo
- **min()** → valor mínimo

Ejemplo:
tupla = (1, 5, 3)
print(len(tupla))
print(max(tupla))
print(min(tupla))

---

10. MÉTODOS DE LAS TUPLAS
Las tuplas tienen pocos métodos debido a su inmutabilidad

- **count()** → cuenta repeticiones
- **index()** → posición de un elemento

Ejemplo:
tupla = (1, 2, 2, 3)
print(tupla.count(2))
print(tupla.index(3))

---

11. INMUTABILIDAD DE LAS TUPLAS
No se pueden modificar

Ejemplo:
tupla = (1, 2, 3)

tupla[0] = 10 → ERROR

---

12. CONVERTIR ENTRE LISTAS Y TUPLAS

Lista a tupla:
lista = [1, 2, 3]
tupla = tuple(lista)

Tupla a lista:
tupla = (1, 2, 3)
lista = list(tupla)

---

13. DESEMPAQUETADO DE TUPLAS
Permite asignar valores a variables

Ejemplo:
tupla = (1, 2, 3)
a, b, c = tupla
print(a, b, c)

---

14. DESEMPAQUETADO CON ASTERISCO
Permite agrupar valores

Ejemplo:
tupla = (1, 2, 3, 4)
a, *b, c = tupla
print(a, b, c)

---

15. CONCATENACIÓN DE TUPLAS

Ejemplo:
a = (1, 2)
b = (3, 4)

```
c = a + b
print(c)
```

---

16. REPETICIÓN DE TUPLAS

Ejemplo:

```
tupla = (1, 2)
print(tupla * 3)
```

---

17. COMPROBAR EXISTENCIA

Ejemplo:

```
tupla = (1, 2, 3)
print(2 in tupla)
```

---

18. TUPLAS ANIDADAS

Ejemplo:

```
tupla = ((1, 2), (3, 4))
print(tupla[0][1])
```

---

19. APLICACIONES PRÁCTICAS DE LAS TUPLAS

1. Coordenadas
   ```
   punto = (10, 20)
   ```
2. Retornar múltiples valores
   ```
   def datos():
   return (1, 2)
   ```
3. Almacenar datos constantes
   ```
   dias = ("lunes", "martes", "miércoles")
   ```

---

20. DIFERENCIAS ENTRE LISTAS Y TUPLAS

Lista:

- Mutable
- Usa []

Tupla:

- Inmutable
- Usa ()

---

21. VENTAJAS DE LAS TUPLAS

- Más rápidas
- Seguras (no cambian)
- Menor uso de memoria

---

22. EJEMPLOS PRÁCTICOS COMPLETOS

Ejemplo 1: almacenar coordenadas

```
punto = (5, 10)
x, y = punto
print(x, y)
```

Ejemplo 2: devolver múltiples valores

```
def operacion(a, b):
return (a+b, a-b)

suma, resta = operacion(5, 3)
print(suma, resta)
```

Ejemplo 3: contar elementos repetidos

```
tupla = (1, 2, 2, 3)
print(tupla.count(2))
```

Ejemplo 4: encontrar índice

```
print(tupla.index(3))
```

---

23. USO EN LA VIDA REAL

- Coordenadas GPS
- Datos que no deben cambiar
- Resultados de funciones
- Configuraciones

---

24. ERRORES COMUNES

- Olvidar la coma en tuplas de un elemento
- Intentar modificar una tupla
- Confundir lista con tupla

---

25. RESUMEN FINAL

- Las **tuplas son estructuras inmutables**
- Mantienen el orden
- Permiten duplicados

- Son eficientes y seguras
- Se usan cuando los datos no deben cambiar

Esta lección incluye:

- Conceptos fundamentales
- Ejemplos paso a paso
- Uso práctico real
- Operaciones completas
- Buenas prácticas

---

1. Crear una tupla e imprimir sus elementos

```
tupla = (1, 2, 3)
for x in tupla:
    print(x)
```

2. Acceder a un elemento por índice

```
tupla = (10, 20, 30)
print(tupla[1])
```

3. Acceder al último elemento

```
tupla = (5, 10, 15)
print(tupla[-1])
```

4. Obtener la longitud de una tupla

```
tupla = (1, 2, 3, 4)
print(len(tupla))
```

5. Contar cuántas veces aparece un elemento

```
tupla = (1, 2, 2, 3)
print(tupla.count(2))
```

6. Encontrar el índice de un valor

```
tupla = (5, 10, 15)
print(tupla.index(10))
```

7. Desempaquetar una tupla

```
tupla = (1, 2, 3)
a, b, c = tupla
print(a, b, c)
```

8. Desempaquetado con asterisco

```
tupla = (1, 2, 3, 4)
a, *b, c = tupla
print(a, b, c)
```

9. Sumar elementos de una tupla

```
tupla = (1, 2, 3)
print(sum(tupla))
```

10. Encontrar valor máximo y mínimo

```
tupla = (5, 2, 9)
print(max(tupla), min(tupla))
```

---

11. Convertir lista a tupla

```
lista = [1, 2, 3]
tupla = tuple(lista)
print(tupla)
```

12. Convertir tupla a lista

```
tupla = (1, 2, 3)
lista = list(tupla)
print(lista)
```

13. Concatenar dos tuplas

```
a = (1, 2)
b = (3, 4)
print(a + b)
```

14. Repetir una tupla

```
tupla = (1, 2)
print(tupla * 3)
```

15. Verificar si un elemento existe

```
tupla = (1, 2, 3)
print(2 in tupla)
```

16. Obtener sub-tupla (slicing)

```
tupla = (1, 2, 3, 4, 5)
print(tupla[1:4])
```

17. Crear tupla de un solo elemento

```
tupla = (5,)
print(tupla)
```

18. Crear tupla desde input

```
datos = tuple(int(input("Número: ")) for _ in range(3))
print(datos)
```

19. Calcular promedio de una tupla

```
tupla = (10, 20, 30)
print(sum(tupla)/len(tupla))
```

20. Contar números pares en una tupla

```
tupla = (1, 2, 3, 4)
pares = sum(1 for x in tupla if x % 2 == 0)
print(pares)
```

---

21. Crear tupla de cuadrados

```
tupla = tuple(x**2 for x in range(5))
print(tupla)
```

22. Filtrar números mayores a 5

```
tupla = (1, 6, 3, 8)
resultado = tuple(x for x in tupla if x > 5)
print(resultado)
```

23. Invertir una tupla

```
tupla = (1, 2, 3)
print(tupla[::-1])
```

24. Encontrar elementos únicos

```
tupla = (1, 2, 2, 3)
print(tuple(set(tupla)))
```

25. Ordenar una tupla

```
tupla = (3, 1, 2)
print(tuple(sorted(tupla)))
```

26. Crear tupla con números negativos

```
tupla = (-1, 2, -3, 4)
negativos = tuple(x for x in tupla if x < 0)
print(negativos)
```

27. Multiplicar todos los elementos

```
tupla = (2, 3, 4)
resultado = 1
for x in tupla:
    resultado *= x
print(resultado)
```

28. Encontrar el segundo mayor número

```
tupla = (1, 5, 3, 4)
print(sorted(tupla)[-2])
```

29. Eliminar duplicados manteniendo orden

```
tupla = (1, 2, 2, 3)
resultado = tuple(dict.fromkeys(tupla))
print(resultado)
```

30. Combinar dos tuplas sin duplicados

```
a = (1, 2)
b = (2, 3)
print(tuple(set(a + b)))
```

---

31. Crear tupla de vocales en una palabra

```
texto = "python"
vocales = tuple(x for x in texto if x in "aeiou")
print(vocales)
```

32. Contar letras en una tupla

```
tupla = ("a", "b", "a")
print(tupla.count("a"))
```

33. Verificar si todas son positivas

```
tupla = (1, 2, 3)
print(all(x > 0 for x in tupla))
```

34. Verificar si existe algún negativo

```
tupla = (1, -2, 3)
print(any(x < 0 for x in tupla))
```

35. Crear tupla de múltiplos de 3

```
tupla = tuple(x for x in range(1, 20) if x % 3 == 0)
print(tupla)
```

36. Sumar dos tuplas elemento a elemento

```
a = (1, 2, 3)
b = (4, 5, 6)
resultado = tuple(x+y for x, y in zip(a, b))
print(resultado)
```

37. Encontrar coincidencias entre tuplas

```
a = (1, 2, 3)
b = (2, 3, 4)
print(tuple(set(a) & set(b)))
```

38. Crear tupla de longitud fija con ceros

```
tupla = (0,) * 5
print(tupla)
```

39. Convertir cadena a tupla de caracteres

```
texto = "hola"
print(tuple(texto))
```

40. Obtener índices de un valor repetido

```
tupla = (1, 2, 2, 3)
indices = [i for i, x in enumerate(tupla) if x == 2]
print(indices)
```

---

41. Crear tupla de números aleatorios (simples)

```
import random
tupla = tuple(random.randint(1, 10) for _ in range(5))
print(tupla)
```

42. Contar números mayores al promedio

```
tupla = (10, 20, 30)
prom = sum(tupla)/len(tupla)
print(sum(1 for x in tupla if x > prom))
```

43. Separar pares e impares

```
tupla = (1, 2, 3, 4)
pares = tuple(x for x in tupla if x % 2 == 0)
impares = tuple(x for x in tupla if x % 2 != 0)
print(pares, impares)
```

44. Convertir tupla a string

```
tupla = ('a', 'b', 'c')
print("".join(tupla))
```

45. Crear tupla de palabras únicas

```
texto = "hola hola mundo"
print(tuple(set(texto.split())))
```

46. Encontrar palabra más larga

```
tupla = ("hola", "programacion", "python")
print(max(tupla, key=len))
```

47. Contar caracteres totales

```
tupla = ("hola", "python")
print(sum(len(x) for x in tupla))
```

48. Crear tupla con valores absolutos

```
tupla = (-1, -2, 3)
print(tuple(abs(x) for x in tupla))
```

49. Generar tupla de potencias de 2

```
tupla = tuple(2**x for x in range(5))
print(tupla)
```

50. Intercambiar valores usando tuplas

```
a = 5
b = 10
a, b = b, a
print(a, b)
```

---

7. LECCIÓN: ESTRUCTURAS DE DATOS EN PYTHON – DICCIONARIOS

1. ¿QUÉ ES UN DICCIONARIO EN PYTHON?

Un **diccionario (dictionary)** es una estructura de datos que almacena información en pares **clave:valor**.

Cada elemento del diccionario tiene:

- Una **clave (key)**
- Un **valor (value)**

Ejemplo:
diccionario = {"nombre": "Juan", "edad": 25}
print(diccionario)

---

2. CARACTERÍSTICAS IMPORTANTES DE LOS DICCIONARIOS

- Son **mutables (se pueden modificar)**
- No permiten **claves duplicadas**
- Los valores sí pueden repetirse
- Son **no ordenados** (aunque en versiones recientes mantienen el orden de inserción)

---

3. CREACIÓN DE DICCIONARIOS

Con llaves {}:
persona = {"nombre": "Ana", "edad": 20}

Con dict():
persona = dict(nombre="Ana", edad=20)

---

4. ACCESO A LOS VALORES

Se accede mediante la clave

Ejemplo:
persona = {"nombre": "Ana", "edad": 20}
print(persona["nombre"])

Importante: si la clave no existe, genera error

---

5. MÉTODO get()

Permite acceder sin error

Ejemplo:
print(persona.get("nombre"))
print(persona.get("direccion", "No existe"))

---

6. MODIFICAR VALORES

Ejemplo:
persona["edad"] = 21

---

7. AGREGAR ELEMENTOS

Ejemplo:
persona["ciudad"] = "Madrid"

---

8. ELIMINAR ELEMENTOS

Métodos:

- del
- pop()
- popitem()
- clear()

Ejemplo:
del persona["edad"]
persona.pop("nombre")
persona.clear()

---

9. RECORRER UN DICCIONARIO

Recorrer claves:
for clave in persona:
print(clave)

Recorrer valores:
for valor in persona.values():
print(valor)

Recorrer ambos:
for clave, valor in persona.items():
print(clave, valor)

---

10. MÉTODOS PRINCIPALES

- keys() → devuelve claves
- values() → devuelve valores
- items() → devuelve pares

Ejemplo:
print(persona.keys())
print(persona.values())

---

11. VERIFICAR EXISTENCIA DE CLAVES

Ejemplo:
print("nombre" in persona)

---

12. COPIAR DICCIONARIOS

Ejemplo:
copia = persona.copy()

---

13. DICCIONARIOS ANIDADOS

Ejemplo:
personas = {
"persona1": {"nombre": "Ana", "edad": 20},
"persona2": {"nombre": "Luis", "edad": 30}
}

---

14. DICCIONARIOS CON LISTAS Y TUPLAS

Ejemplo:
datos = {"numeros": [1, 2, 3], "coordenadas": (10, 20)}

---

15. CREAR DICCIONARIO CON BUCLES (COMPRENSIÓN)

Ejemplo:
cuadrados = {x: x**2 for x in range(5)}
print(cuadrados)

---

16. ACTUALIZAR DICCIONARIOS

Ejemplo:
a = {"a": 1}
b = {"b": 2}
a.update(b)
print(a)

---

17. ELIMINAR TODOS LOS ELEMENTOS

Ejemplo:
persona.clear()

---

18. LONGITUD DE UN DICCIONARIO

Ejemplo:
print(len(persona))

---

19. APLICACIONES PRÁCTICAS

1. Almacenar datos de usuarios
2. Contar elementos
3. Representar JSON
4. Base de datos simple

---

20. CONTAR FRECUENCIA DE ELEMENTOS

Ejemplo:
texto = "hola mundo"
frecuencia = {}

for letra in texto:
frecuencia[letra] = frecuencia.get(letra, 0) + 1

print(frecuencia)

---

21. ORDENAR DICCIONARIOS

Por clave:
print(dict(sorted(persona.items())))

Por valor:
print(sorted(persona.items(), key=lambda x: x[1]))

---

22. FUSIONAR DICCIONARIOS

Ejemplo:
a = {"a": 1}
b = {"b": 2}
c = {**a, **b}
print(c)

---

23. FILTRAR DICCIONARIOS

Ejemplo:
dic = {"a": 1, "b": 2, "c": 3}
filtrado = {k: v for k, v in dic.items() if v > 1}
print(filtrado)

---

24. CONVERTIR LISTAS A DICCIONARIO

Ejemplo:
claves = ["a", "b"]
valores = [1, 2]
dic = dict(zip(claves, valores))
print(dic)

---

25. EJEMPLOS PRÁCTICOS COMPLETOS

Ejemplo 1: agenda de contactos
agenda = {}
agenda["Ana"] = "12345"
agenda["Luis"] = "67890"
print(agenda)

Ejemplo 2: inventario
inventario = {"manzana": 10, "pera": 5}
inventario["manzana"] += 5
print(inventario)

Ejemplo 3: contar palabras
texto = "hola hola mundo"
palabras = texto.split()
contador = {}

for palabra in palabras:
contador[palabra] = contador.get(palabra, 0) + 1

print(contador)

---

26. ERRORES COMUNES

- Usar claves duplicadas
- Acceder a claves inexistentes
- Confundir listas con diccionarios

---

27. VENTAJAS DE LOS DICCIONARIOS

- Acceso rápido
- Estructura flexible

- Fácil organización de datos
- Muy usados en aplicaciones reales

---

28. DIFERENCIAS CON OTRAS ESTRUCTURAS

Lista → ordenada, índices numéricos
Tupla → inmutable
Conjunto → sin duplicados
Diccionario → clave:valor

---

29. BUENAS PRÁCTICAS

- Usar claves descriptivas
- Evitar claves duplicadas
- Usar get() para evitar errores
- Mantener estructura clara

---

30. RESUMEN FINAL

- Los **diccionarios almacenan datos en pares clave:valor**
- Son **mutables y muy eficientes**
- Permiten organizar información compleja
- Son fundamentales en programación

Esta lección incluye:

- Conceptos básicos y avanzados
- Métodos y operaciones
- Ejemplos reales
- Aplicaciones prácticas
- Buenas prácticas

---

1. Crear un diccionario e imprimirlo

```
dic = {"nombre": "Ana", "edad": 20}
print(dic)
```

2. Acceder a un valor por clave

```
dic = {"nombre": "Ana"}
print(dic["nombre"])
```

3. Usar get() para evitar errores

```
dic = {"nombre": "Ana"}
print(dic.get("edad", "No existe"))
```

4. Agregar un nuevo elemento

```
dic = {}
dic["ciudad"] = "Madrid"
print(dic)
```

5. Modificar un valor

```
dic = {"edad": 20}
dic["edad"] = 25
print(dic)
```

6. Eliminar un elemento con del

```
dic = {"a": 1, "b": 2}
del dic["a"]
print(dic)
```

7. Eliminar con pop()

```
dic = {"a": 1, "b": 2}
dic.pop("b")
print(dic)
```

8. Recorrer claves

```
dic = {"a": 1, "b": 2}
for k in dic:
    print(k)
```

9. Recorrer valores

```
dic = {"a": 1, "b": 2}
for v in dic.values():
    print(v)
```

10. Recorrer clave y valor

```
dic = {"a": 1, "b": 2}
for k, v in dic.items():
    print(k, v)
```

---

11. Verificar si existe una clave

```
dic = {"a": 1}
print("a" in dic)
```

12. Longitud del diccionario

```
dic = {"a": 1, "b": 2}
print(len(dic))
```

13. Copiar un diccionario

```
dic = {"a": 1}
copia = dic.copy()
print(copia)
```

14. Vaciar un diccionario

```
dic = {"a": 1}
dic.clear()
print(dic)
```

15. Crear diccionario con comprensión

```
dic = {x: x**2 for x in range(5)}
print(dic)
```

16. Contar letras en una cadena

```
texto = "hola"
conteo = {}
for letra in texto:
    conteo[letra] = conteo.get(letra, 0) + 1
print(conteo)
```

17. Contar palabras

```
texto = "hola hola mundo"
dic = {}
for palabra in texto.split():
    dic[palabra] = dic.get(palabra, 0) + 1
print(dic)
```

18. Sumar valores del diccionario

```
dic = {"a": 1, "b": 2}
print(sum(dic.values()))
```

19. Obtener clave con valor máximo

```
dic = {"a": 1, "b": 5}
print(max(dic, key=dic.get))
```

20. Obtener valor máximo

```
dic = {"a": 1, "b": 5}
print(max(dic.values()))
```

---

21. Invertir claves y valores

```
dic = {"a": 1, "b": 2}
invertido = {v: k for k, v in dic.items()}
print(invertido)
```

22. Fusionar diccionarios

```
a = {"a": 1}
b = {"b": 2}
c = {**a, **b}
print(c)
```

23. Actualizar diccionario

```
dic = {"a": 1}
dic.update({"b": 2})
print(dic)
```

24. Filtrar valores mayores a 2

```
dic = {"a": 1, "b": 3}
print({k: v for k, v in dic.items() if v > 2})
```

25. Ordenar por claves

```
dic = {"b": 2, "a": 1}
print(dict(sorted(dic.items())))
```

26. Ordenar por valores

```
dic = {"a": 3, "b": 1}
print(sorted(dic.items(), key=lambda x: x[1]))
```

27. Crear diccionario desde listas

```
claves = ["a", "b"]
valores = [1, 2]
print(dict(zip(claves, valores)))
```

28. Verificar si todos los valores son positivos

```
dic = {"a": 1, "b": 2}
print(all(v > 0 for v in dic.values()))
```

29. Verificar si hay algún valor negativo

```
dic = {"a": 1, "b": -2}
print(any(v < 0 for v in dic.values()))
```

30. Crear diccionario con números pares

```
dic = {x: x for x in range(10) if x % 2 == 0}
print(dic)
```

---

31. Crear agenda de contactos

```
agenda = {}
agenda["Ana"] = "123"
agenda["Luis"] = "456"
print(agenda)
```

32. Inventario con actualización

```
inv = {"manzana": 10}
inv["manzana"] += 5
print(inv)
```

33. Eliminar elemento con popitem()

```
dic = {"a": 1, "b": 2}
dic.popitem()
print(dic)
```

34. Obtener todas las claves

```
dic = {"a": 1, "b": 2}
print(dic.keys())
```

35. Obtener todos los valores

```
dic = {"a": 1, "b": 2}
print(dic.values())
```

36. Obtener pares clave-valor

```
dic = {"a": 1}
print(dic.items())
```

37. Diccionario anidado

```
dic = {"persona": {"nombre": "Ana", "edad": 20}}
print(dic["persona"]["nombre"])
```

38. Sumar valores de diccionario anidado

```
dic = {"a": {"v": 1}, "b": {"v": 2}}
print(sum(x["v"] for x in dic.values()))
```

39. Crear diccionario con cuadrados

```
print({x: x**2 for x in range(1, 6)})
```

40. Convertir diccionario a lista de claves

```
dic = {"a": 1, "b": 2}
print(list(dic.keys()))
```

---

41. Convertir diccionario a lista de valores

```
dic = {"a": 1, "b": 2}
print(list(dic.values()))
```

42. Eliminar claves con valor menor a 2

```
dic = {"a": 1, "b": 3}
dic = {k: v for k, v in dic.items() if v >= 2}
print(dic)
```

43. Multiplicar valores por 2

```
dic = {"a": 1, "b": 2}
print({k: v*2 for k, v in dic.items()})
```

44. Encontrar clave con valor mínimo

```
dic = {"a": 5, "b": 1}
print(min(dic, key=dic.get))
```

45. Crear diccionario con caracteres ASCII

```
dic = {chr(x): x for x in range(65, 70)}
print(dic)
```

46. Contar números en lista usando diccionario

```
lista = [1, 2, 2, 3]
dic = {}
for x in lista:
    dic[x] = dic.get(x, 0) + 1
print(dic)
```

47. Unir diccionarios sumando valores

```
a = {"a": 1}
b = {"a": 2}
resultado = {}
for k in set(a) | set(b):
    resultado[k] = a.get(k, 0) + b.get(k, 0)
print(resultado)
```

48. Crear diccionario desde input

```
dic = {}
for _ in range(3):
    k = input("Clave: ")
    v = input("Valor: ")
    dic[k] = v
print(dic)
```

49. Crear diccionario con índices de lista

```
lista = ["a", "b", "c"]
dic = {i: lista[i] for i in range(len(lista))}
print(dic)
```

50. Intercambiar claves y valores (valores únicos)

```
dic = {"a": 1, "b": 2}
print({v: k for k, v in dic.items()})
```

---

8. LECCIÓN: FUNCIONES EN PYTHON

1. ¿QUÉ ES UNA FUNCIÓN?

Una **función** es un bloque de código que realiza una tarea específica y puede ser reutilizado varias veces dentro de un programa.

Permite:

- Organizar el código
- Evitar repetir instrucciones
- Mejorar la claridad

---

2. DEFINICIÓN DE UNA FUNCIÓN

Se define usando la palabra clave **def**

Ejemplo:

```
def saludar():
print("Hola")
```

Para ejecutar la función:

```
saludar()
```

---

3. PARTES DE UNA FUNCIÓN

- **Nombre de la función**
- **Parámetros (opcional)**
- **Cuerpo de la función**
- **Valor de retorno (opcional)**

---

4. FUNCIONES SIN PARÁMETROS

Ejemplo:

```
def mensaje():
print("Bienvenido")

mensaje()
```

---

5. FUNCIONES CON PARÁMETROS

Los parámetros permiten recibir datos

Ejemplo:

```
def saludar(nombre):
print("Hola", nombre)

saludar("Ana")
```

---

6. FUNCIONES CON VARIOS PARÁMETROS

Ejemplo:
def suma(a, b):
print(a + b)

suma(3, 5)

---

7. VALOR DE RETORNO (RETURN)
Permite devolver un resultado

Ejemplo:
def suma(a, b):
return a + b

resultado = suma(3, 4)
print(resultado)

---

8. DIFERENCIA ENTRE PRINT Y RETURN

- **print()** muestra el resultado
- **return** devuelve el valor

---

9. PARÁMETROS POR DEFECTO

Ejemplo:
def saludar(nombre="Usuario"):
print("Hola", nombre)

saludar()
saludar("Luis")

---

10. ARGUMENTOS POSICIONALES Y NOMBRADOS

Ejemplo:
def persona(nombre, edad):
print(nombre, edad)

persona("Ana", 20)
persona(edad=25, nombre="Luis")

---

11. FUNCIONES CON RETORNO MÚLTIPLE

Ejemplo:
def operaciones(a, b):
return a+b, a-b

suma, resta = operaciones(5, 3)

---

12. FUNCIONES ANIDADAS

Ejemplo:
def externa():
def interna():
print("Dentro")
interna()

externa()

---

13. ÁMBITO DE VARIABLES (SCOPE)

- **Variable local** → dentro de la función
- **Variable global** → fuera

Ejemplo:
x = 10

def mostrar():
x = 5
print(x)

mostrar()
print(x)

---

14. USO DE GLOBAL

Ejemplo:
x = 10

def cambiar():
global x
x = 20

cambiar()
print(x)

---

15. FUNCIONES RECURSIVAS
Una función que se llama a sí misma

Ejemplo:
def factorial(n):
if n == 1:
return 1
return n * factorial(n-1)

---

16. FUNCIONES LAMBDA
Funciones pequeñas en una sola línea

Ejemplo:
suma = lambda a, b: a + b
print(suma(2, 3))

**17. PARÁMETROS ARBITRARIOS (args)*

Ejemplo:
def suma(*numeros):
return sum(numeros)

print(suma(1, 2, 3))

---

18. PARÁMETROS CON CLAVE (kwargs)**

Ejemplo:
def datos(**info):
print(info)

datos(nombre="Ana", edad=20)

---

19. DOCUMENTACIÓN DE FUNCIONES

Ejemplo:
def suma(a, b):
"""Suma dos números"""
return a + b

---

20. FUNCIONES COMO PARÁMETROS

Ejemplo:
def aplicar(func, a, b):
return func(a, b)

def suma(a, b):
return a + b

print(aplicar(suma, 2, 3))

---

21. FUNCIONES QUE DEVUELVEN FUNCIONES

Ejemplo:
def externa():
def interna():

```
return "Hola"
return interna

f = externa()
print(f())
```

---

22. FUNCIONES CON LISTAS

Ejemplo:

```
def promedio(lista):
return sum(lista)/len(lista)

print(promedio([1,2,3]))
```

---

23. FUNCIONES CON DICCIONARIOS

Ejemplo:

```
def mostrar(dic):
for k, v in dic.items():
print(k, v)
```

---

24. BUENAS PRÁCTICAS

- Usar nombres claros
- Evitar funciones muy largas
- Reutilizar código
- Documentar funciones

---

25. EJEMPLOS PRÁCTICOS COMPLETOS

Ejemplo 1: calcular área de un círculo

```
def area(r):
return 3.14 * r**2

print(area(5))
```

Ejemplo 2: verificar número par

```
def es_par(n):
return n % 2 == 0

print(es_par(4))
```

Ejemplo 3: contar letras en texto

```
def contar(texto):
return len(texto)

print(contar("hola"))
```

Ejemplo 4: máximo de una lista

```
def maximo(lista):
return max(lista)
```

Ejemplo 5: convertir grados

```
def celsius_a_fahrenheit(c):
return (c * 9/5) + 32
```

---

26. ERRORES COMUNES

- Olvidar return
- Mal uso de parámetros
- Variables fuera de alcance
- Llamar funciones sin definir

---

27. VENTAJAS DE LAS FUNCIONES

- Reutilización
- Organización
- Mantenimiento fácil
- Código más limpio

---

28. DIFERENCIA ENTRE FUNCIÓN Y PROCEDIMIENTO

- Función → devuelve valor
- Procedimiento → no devuelve

---

29. RESUMEN FINAL

- Las **funciones permiten dividir el programa en partes reutilizables**
- Se definen con **def**
- Pueden tener **parámetros y retorno**
- Existen funciones avanzadas como **lambda, recursivas, args y kwargs**

- Son fundamentales en cualquier programa

Esta lección incluye:

- Conceptos básicos y avanzados
- Ejemplos completos
- Aplicaciones reales
- Buenas prácticas

---

1. Crear una función que imprima un saludo

```
def saludo():
    print("Hola, bienvenido")
saludo()
```

2. Función que reciba un nombre y lo imprima

```
def saludar(nombre):
    print("Hola", nombre)
saludar("Ana")
```

3. Función que sume dos números y muestre el resultado

```
def suma(a, b):
    print("La suma es:", a + b)
suma(5, 3)
```

4. Función que devuelva la suma de dos números

```
def suma(a, b):
    return a + b
resultado = suma(4, 6)
print(resultado)
```

5. Función con parámetros por defecto

```
def saludar(nombre="Usuario"):
    print("Hola", nombre)
saludar()
saludar("Luis")
```

6. Función con varios parámetros

```
def datos(nombre, edad):
    print("Nombre:", nombre, "Edad:", edad)
datos("Ana", 25)
```

7. Función que devuelva el cuadrado de un número

```
def cuadrado(n):
    return n**2
print(cuadrado(5))
```

8. Función que devuelva el mayor de dos números

```
def mayor(a, b):
    if a > b:
        return a
    else:
        return b
print(mayor(10, 7))
```

9. Función que verifique si un número es par

```
def es_par(n):
    return n % 2 == 0
print(es_par(4))
```

10. Función que reciba una lista y devuelva su promedio

```
def promedio(lista):
    return sum(lista)/len(lista)
print(promedio([1, 2, 3, 4]))
```

---

11. Función que reciba una cadena y devuelva su longitud

```
def longitud(texto):
    return len(texto)
print(longitud("Python"))
```

12. Función que convierta grados Celsius a Fahrenheit

```
def celsius_a_fahrenheit(c):
    return (c * 9/5) + 32
print(celsius_a_fahrenheit(25))
```

13. Función que convierta Fahrenheit a Celsius

```
def fahrenheit_a_celsius(f):
    return (f - 32) * 5/9
print(fahrenheit_a_celsius(77))
```

14. Función que reciba un número y devuelva su factorial

```
def factorial(n):
    if n == 1 or n == 0:
        return 1
    return n * factorial(n-1)
print(factorial(5))
```

15. Función que reciba una lista y devuelva el máximo

```
def maximo(lista):
    return max(lista)
print(maximo([2, 5, 9, 1]))
```

16. Función que reciba una lista y devuelva el mínimo

```
def minimo(lista):
    return min(lista)
print(minimo([2, 5, 9, 1]))
```

17. Función que reciba una lista y devuelva los números pares

```
def pares(lista):
    return [x for x in lista if x % 2 == 0]
print(pares([1, 2, 3, 4]))
```

18. Función que reciba una lista y devuelva los números impares

```
def impares(lista):
    return [x for x in lista if x % 2 != 0]
print(impares([1, 2, 3, 4]))
```

19. Función que reciba un número y devuelva True si es primo

```
def es_primo(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
print(es_primo(7))
```

20. Función que reciba un texto y cuente las vocales

```
def contar_vocales(texto):
    return sum(1 for x in texto.lower() if x in "aeiou")
print(contar_vocales("Python"))
```

---

21. Función que reciba una lista y devuelva su suma

```
def suma_lista(lista):
    return sum(lista)
print(suma_lista([1, 2, 3, 4]))
```

22. Función que reciba un número y genere la tabla de multiplicar

```
def tabla(n):
    for i in range(1, 11):
        print(n, "x", i, "=", n*i)
tabla(5)
```

23. Función que reciba una lista y devuelva los números al cuadrado

```
def cuadrados(lista):
    return [x**2 for x in lista]
print(cuadrados([1, 2, 3]))
```

24. Función que reciba dos listas y devuelva la suma elemento a elemento

```
def suma_listas(a, b):
    return [x+y for x, y in zip(a, b)]
print(suma_listas([1,2],[3,4]))
```

25. Función que reciba un diccionario y lo muestre

```
def mostrar_dic(dic):
    for k, v in dic.items():
        print(k, ":", v)
mostrar_dic({"nombre":"Ana","edad":20})
```

26. Función que reciba un diccionario y devuelva la clave con valor máximo

```
def clave_max(dic):
    return max(dic, key=dic.get)
print(clave_max({"a":1,"b":5,"c":3}))
```

27. Función que reciba un diccionario y devuelva la clave con valor mínimo

```
def clave_min(dic):
    return min(dic, key=dic.get)
print(clave_min({"a":1,"b":5,"c":3}))
```

28. Función que reciba un diccionario y devuelva la suma de sus valores

```
def suma_valores(dic):
    return sum(dic.values())
print(suma_valores({"a":1,"b":2,"c":3}))
```

29. Función que reciba un texto y devuelva las palabras

```
def separar_palabras(texto):
    return texto.split()
print(separar_palabras("Python es divertido"))
```

30. Función que reciba una lista de palabras y devuelva las más largas

```
def palabra_larga(lista):
    return max(lista, key=len)
print(palabra_larga(["Python","es","divertido"]))
```

---

31. Función recursiva que calcule la serie de Fibonacci

```
def fibonacci(n):
    if n == 0: return 0
    if n == 1: return 1
    return fibonacci(n-1) + fibonacci(n-2)
print(fibonacci(6))
```

32. Función que reciba una lista y elimine duplicados

```
def sin_duplicados(lista):
    return list(dict.fromkeys(lista))
print(sin_duplicados([1,2,2,3]))
```

33. Función que reciba una lista y devuelva la lista invertida

```
def invertir(lista):
    return lista[::-1]
print(invertir([1,2,3]))
```

34. Función lambda para multiplicar dos números

```
multiplicar = lambda a, b: a * b
print(multiplicar(3,4))
```

35. Función lambda para verificar paridad

```
es_par = lambda n: n%2==0
print(es_par(5))
```

**36. Función que reciba varios números (args) y devuelva la suma*

```
def suma_varios(*nums):
    return sum(nums)
print(suma_varios(1,2,3,4))
```

37. Función que reciba varios parámetros con nombre (kwargs) y los muestre**

```
def mostrar_info(**info):
    for k, v in info.items():
        print(k, ":", v)
mostrar_info(nombre="Ana", edad=20)
```

38. Función que reciba una lista de números y devuelva los pares multiplicados por 2

```
def pares_dobles(lista):
    return [x*2 for x in lista if x%2==0]
print(pares_dobles([1,2,3,4]))
```

39. Función que reciba un número y devuelva True si es positivo

```
def es_positivo(n):
    return n>0
print(es_positivo(-2))
```

40. Función que reciba una lista y devuelva los elementos mayores que 5

```
def mayores_cinco(lista):
    return [x for x in lista if x>5]
print(mayores_cinco([2,6,8,1]))
```

---

41. Función que reciba un diccionario y elimine claves con valor 0

```
def limpiar(dic):
    return {k:v for k,v in dic.items() if v!=0}
print(limpiar({"a":0,"b":2,"c":0}))
```

42. Función que reciba dos listas y devuelva elementos comunes

```
def comunes(a,b):
    return [x for x in a if x in b]
print(comunes([1,2,3],[2,3,4]))
```

43. Función que reciba una cadena y devuelva en mayúsculas

```
def mayusculas(texto):
    return texto.upper()
print(mayusculas("hola"))
```

44. Función que reciba una cadena y devuelva en minúsculas

```
def minusculas(texto):
    return texto.lower()
print(minusculas("HOLA"))
```

45. Función que reciba un número y devuelva True si es divisible entre 3

```
def divisible_3(n):
    return n%3==0
print(divisible_3(9))
```

46. Función que reciba una lista y devuelva los números negativos

```
def negativos(lista):
    return [x for x in lista if x<0]
print(negativos([1,-2,3,-4]))
```

47. Función que reciba un número y devuelva su valor absoluto

```
def absoluto(n):
    return abs(n)
print(absoluto(-5))
```

48. Función que reciba un número y devuelva True si es múltiplo de 5

```
def multiplo_5(n):
    return n%5==0
print(multiplo_5(20))
```

49. Función que reciba una lista y devuelva los elementos únicos

```
def unicos(lista):
    return list(set(lista))
print(unicos([1,2,2,3,3,4]))
```

50. Función que reciba una lista y devuelva el segundo mayor número

```
def segundo_mayor(lista):
    return sorted(lista)[-2]
print(segundo_mayor([1,5,2,4]))
```

---

9. LECCIÓN: PROGRAMACIÓN ORIENTADA A OBJETOS EN PYTHON

---

1. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

La **Programación Orientada a Objetos** es un paradigma de programación que organiza el código en **objetos**, los cuales combinan **datos** y **funciones**. Python es un lenguaje que soporta POO de manera nativa.

- **Objeto:** Entidad que combina **atributos** (datos) y **métodos** (funciones que actúan sobre los datos).
- **Clase:** Plantilla o molde para crear objetos. Define los atributos y métodos que los objetos tendrán.
- **Instancia:** Cada objeto creado a partir de una clase.
- **Encapsulamiento:** Ocultar la complejidad interna de los objetos y exponer solo lo necesario.
- **Herencia:** Permite crear nuevas clases a partir de clases existentes, heredando sus atributos y métodos.
- **Polimorfismo:** Capacidad de un objeto de comportarse de diferentes formas dependiendo del contexto.
- **Abstracción:** Representación de conceptos esenciales sin incluir detalles innecesarios.

---

2. CREACIÓN DE CLASES Y OBJETOS

En Python, se define una clase usando la palabra clave `class`.

Ejemplo básico de clase:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # Atributo
        self.edad = edad      # Atributo

    def saludar(self):       # Método
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")

# Crear una instancia de Persona
persona1 = Persona("Ana", 25)
persona1.saludar()
```

Explicación:

- `__init__` es el **método constructor**, se ejecuta automáticamente al crear una instancia.
- `self` representa la **propia instancia del objeto** y permite acceder a sus atributos y métodos.

---

3. ATRIBUTOS Y MÉTODOS

- **Atributos:** Variables dentro de un objeto.
- **Métodos:** Funciones dentro de un objeto.

Tipos de atributos:

- **Atributos de instancia:** Propios de cada objeto (`self.nombre`).
- **Atributos de clase:** Compartidos entre todas las instancias (`Clase.atributo`).

Ejemplo de atributos de clase:

```
class Persona:
    especie = "Humano"  # Atributo de clase

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

persona1 = Persona("Luis", 30)
persona2 = Persona("Marta", 22)

print(persona1.especie)  # Humano
print(persona2.especie)  # Humano
```

---

4. ENCAPSULAMIENTO

El **encapsulamiento** protege los datos y evita modificaciones directas no deseadas.

- **Público:** Se puede acceder desde cualquier lugar.
- **Protegido:** Se indica con un guion bajo `_atributo`. Convención para uso interno.
- **Privado:** Se indica con doble guion bajo `__atributo`. Acceso restringido.

Ejemplo:

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.__saldo = saldo  # Privado

    def depositar(self, monto):
        if monto > 0:
            self.__saldo += monto

    def mostrar_saldo(self):
        print(f"Saldo: {self.__saldo}")

cuenta = CuentaBancaria("Carlos", 1000)
cuenta.depositar(500)
cuenta.mostrar_saldo()
```

---

5. HERENCIA

La **herencia** permite crear clases que reutilizan atributos y métodos de otra clase (superclase).

Ejemplo:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        pass  # Método genérico

class Perro(Animal):
    def hablar(self):
        print("Guau")

class Gato(Animal):
    def hablar(self):
        print("Miau")

perro = Perro("Rex")
gato = Gato("Luna")
perro.hablar()  # Guau
gato.hablar()   # Miau
```

Explicación:

- `Perro` y `Gato` **heredan** de `Animal`.
- Se puede **sobrescribir métodos** de la clase base (polimorfismo).

---

6. POLIMORFISMO

El polimorfismo permite que diferentes objetos respondan al mismo método de manera distinta.

Ejemplo:

```
def hacer_hablar(animal):
    animal.hablar()

hacer_hablar(perro)  # Guau
hacer_hablar(gato)   # Miau
```

---

7. MÉTODOS ESPECIALES EN PYTHON

Python incluye métodos especiales para mejorar la funcionalidad de las clases:

- `__str__`: Define cómo se muestra el objeto al usar `print()`.
- `__repr__`: Representación oficial del objeto.
- `__len__`: Permite usar `len()` en objetos.
- `__eq__`: Define cómo se comparan objetos con `==`.

Ejemplo:

```
class Libro:
    def __init__(self, titulo, paginas):
        self.titulo = titulo
        self.paginas = paginas

    def __str__(self):
        return f"{self.titulo}, {self.paginas} páginas"
```

```
libro = Libro("Python Avanzado", 300)
print(libro)  # Python Avanzado, 300 páginas
```

---

8. CLASES Y MÉTODOS ESTÁTICOS

- **Método estático:** No necesita la instancia para funcionar. Se declara con `@staticmethod`.
- **Método de clase:** Recibe la clase como primer argumento (`cls`). Se declara con `@classmethod`.

Ejemplo:

```
class Matematica:
    @staticmethod
    def sumar(a, b):
        return a + b

    @classmethod
    def info(cls):
        print(f"Esta es la clase {cls.__name__}")

print(Matematica.sumar(5, 7))  # 12
Matematica.info()  # Esta es la clase Matematica
```

---

9. COMPOSICIÓN DE OBJETOS

La **composición** permite que un objeto contenga a otros objetos como atributos, creando relaciones "tiene un".

Ejemplo:

```
class Motor:
    def __init__(self, tipo):
        self.tipo = tipo

class Coche:
    def __init__(self, marca, motor):
        self.marca = marca
        self.motor = motor

motor_v8 = Motor("V8")
coche = Coche("Ford", motor_v8)
print(coche.motor.tipo)  # V8
```

---

10. RESUMEN DE CONCEPTOS CLAVE

- **Clase:** Molde para crear objetos.
- **Objeto/Instancia:** Elemento creado a partir de la clase.
- **Atributos y Métodos:** Datos y funciones dentro de objetos.
- **Encapsulamiento:** Protección de datos (`public`, `_protected`, `__private`).

- **Herencia:** Reutilización de código y extensión de clases.
- **Polimorfismo:** Comportamiento diferente según el objeto.
- **Abstracción:** Simplificación del modelo sin mostrar detalles internos.
- **Métodos especiales:** `__init__`, `__str__`, `__repr__`, etc.
- **Métodos estáticos y de clase:** Operaciones que no dependen de la instancia.
- **Composición:** Un objeto contiene otros objetos.

---

1. Programa para crear una clase Persona y mostrar su información

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # **atributo**
        self.edad = edad      # **atributo**

    def mostrar_info(self):  # **método**
        print(f"Nombre: {self.nombre}, Edad: {self.edad}")

persona1 = Persona("Ana", 25)
persona1.mostrar_info()
```

---

2. Programa para calcular el área de un rectángulo usando POO

```
class Rectangulo:
    def __init__(self, base, altura):
        self.base = base  # **atributo**
        self.altura = altura  # **atributo**

    def area(self):  # **método**
        return self.base * self.altura

rect = Rectangulo(5, 10)
print(f"Área: {rect.area()}")
```

---

3. Programa para representar un Coche y mostrar sus detalles

```
class Coche:
    def __init__(self, marca, modelo, año):
        self.marca = marca  # **atributo**
        self.modelo = modelo  # **atributo**
        self.año = año  # **atributo**

    def mostrar_info(self):  # **método**
        print(f"{self.marca} {self.modelo} ({self.año})")

mi_coche = Coche("Toyota", "Corolla", 2020)
mi_coche.mostrar_info()
```

---

4. Programa para representar un Estudiante y calcular promedio de notas

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre  # **atributo**
        self.notas = notas  # **atributo lista**

    def promedio(self):  # **método**
        return sum(self.notas) / len(self.notas)

est = Estudiante("Luis", [8, 9, 7, 10])
print(f"Promedio de {est.nombre}: {est.promedio()}")
```

---

5. Programa para manejar una cuenta bancaria con depósito y retiro

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular  # **atributo**
        self.__saldo = saldo  # **atributo privado**

    def depositar(self, monto):  # **método**
        self.__saldo += monto

    def retirar(self, monto):  # **método**
        if monto <= self.__saldo:
            self.__saldo -= monto
        else:
            print("Saldo insuficiente")

    def mostrar_saldo(self):
        print(f"Saldo de {self.titular}: {self.__saldo}")

cuenta = CuentaBancaria("Carlos", 1000)
cuenta.depositar(500)
cuenta.retirar(300)
cuenta.mostrar_saldo()
```

---

6. Programa para manejar empleados con salario y aumento

```
class Empleado:
    def __init__(self, nombre, salario):
        self.nombre = nombre  # **atributo**
        self.salario = salario  # **atributo**

    def aplicar_aumento(self, porcentaje):  # **método**
        self.salario += self.salario * porcentaje / 100

    def mostrar_info(self):
        print(f"{self.nombre}: ${self.salario:.2f}")

empleado1 = Empleado("Ana", 2000)
empleado1.aplicar_aumento(10)
empleado1.mostrar_info()
```

---

7. Programa para representar un libro con título y autor

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo  # **atributo**
        self.autor = autor    # **atributo**

    def mostrar_info(self):
        print(f"{self.titulo} por {self.autor}")

libro1 = Libro("Python Básico", "Juan Pérez")
libro1.mostrar_info()
```

---

8. Programa para manejar productos en un inventario

```
class Producto:
    def __init__(self, nombre, precio, cantidad):
        self.nombre = nombre  # **atributo**
        self.precio = precio  # **atributo**
        self.cantidad = cantidad  # **atributo**

    def total_inventario(self):
        return self.precio * self.cantidad

prod1 = Producto("Laptop", 1200, 5)
print(f"Total inventario: ${prod1.total_inventario()}")
```

---

9. Programa para calcular la distancia entre dos puntos

```
class Punto:
    def __init__(self, x, y):
        self.x = x  # **atributo**
        self.y = y  # **atributo**

    def distancia(self, otro_punto):  # **método**
        return ((self.x - otro_punto.x)**2 + (self.y - otro_punto.y)**2)**0.5

p1 = Punto(0, 0)
p2 = Punto(3, 4)
print(f"Distancia: {p1.distancia(p2)}")
```

---

10. Programa para manejar una lista de estudiantes y mostrar sus promedios

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre  # **atributo**
        self.notas = notas  # **atributo lista**

    def promedio(self):  # **método**
        return sum(self.notas) / len(self.notas)

estudiantes = [
    Estudiante("Ana", [8, 9, 10]),
    Estudiante("Luis", [7, 6, 8])
]

for est in estudiantes:
    print(f"{est.nombre} - Promedio: {est.promedio()}")
```

---

11. Programa para representar un triángulo y calcular su área

```
class Triangulo:
    def __init__(self, base, altura):
        self.base = base  # **atributo**
        self.altura = altura  # **atributo**

    def area(self):  # **método**
        return (self.base * self.altura) / 2

tri = Triangulo(6, 4)
print(f"Área del triángulo: {tri.area()}")
```

---

12. Programa para representar un círculo y calcular su área y perímetro

```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio  # **atributo**

    def area(self):
        return math.pi * self.radio ** 2  # **método**

    def perimetro(self):
        return 2 * math.pi * self.radio  # **método**

c = Circulo(5)
print(f"Área: {c.area():.2f}, Perímetro: {c.perimetro():.2f}")
```

---

13. Programa para representar una cuenta de ahorros con interés

```
class CuentaAhorros:
    def __init__(self, titular, saldo, interes):
        self.titular = titular  # **atributo**
        self.__saldo = saldo  # **atributo privado**
        self.interes = interes  # **atributo**

    def aplicar_interes(self):  # **método**
        self.__saldo += self.__saldo * self.interes / 100

    def mostrar_saldo(self):
        print(f"Saldo de {self.titular}: {self.__saldo:.2f}")

cuenta = CuentaAhorros("Marta", 1000, 5)
cuenta.aplicar_interes()
cuenta.mostrar_saldo()
```

---

14. Programa para representar un empleado con cálculo de bono

```
class Empleado:
    def __init__(self, nombre, salario):
        self.nombre = nombre  # **atributo**
        self.salario = salario  # **atributo**

    def calcular_bono(self, porcentaje):  # **método**
        return self.salario * porcentaje / 100

emp = Empleado("Jorge", 2500)
print(f"Bono de {emp.nombre}: {emp.calcular_bono(10)}")
```

---

15. Programa para representar un perro y hacer que ladre

```
class Perro:
    def __init__(self, nombre, raza):
        self.nombre = nombre  # **atributo**
        self.raza = raza  # **atributo**

    def ladrar(self):  # **método**
        print(f"{self.nombre} dice: Guau!")

mi_perro = Perro("Rex", "Labrador")
mi_perro.ladrar()
```

---

16. Programa para manejar un inventario de productos y calcular total de todos

```
class Producto:
    def __init__(self, nombre, precio, cantidad):
        self.nombre = nombre  # **atributo**
        self.precio = precio  # **atributo**
        self.cantidad = cantidad  # **atributo**

    def total(self):
        return self.precio * self.cantidad  # **método**

productos = [
    Producto("Mouse", 20, 5),
    Producto("Teclado", 50, 3)
]

total_inventario = sum(p.total() for p in productos)
print(f"Total inventario: ${total_inventario}")
```

---

17. Programa para representar un estudiante y verificar aprobación

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre  # **atributo**
        self.notas = notas  # **atributo lista**

    def promedio(self):  # **método**
        return sum(self.notas) / len(self.notas)

    def aprobado(self):  # **método**
        return self.promedio() >= 6

est = Estudiante("Laura", [5, 7, 8])
print(f"{est.nombre} aprobado: {est.aprobado()}")
```

---

18. Programa para manejar un vehículo y calcular velocidad promedio

```
class Vehiculo:
    def __init__(self, marca, distancia, tiempo):
        self.marca = marca  # **atributo**
        self.distancia = distancia  # **atributo**
        self.tiempo = tiempo  # **atributo**

    def velocidad_promedio(self):  # **método**
        return self.distancia / self.tiempo

v = Vehiculo("Ford", 120, 2)
print(f"Velocidad promedio: {v.velocidad_promedio()} km/h")
```

---

19. Programa para representar un alumno y sus asignaturas

```
class Alumno:
    def __init__(self, nombre):
        self.nombre = nombre  # **atributo**
        self.asignaturas = []  # **atributo lista**

    def agregar_asignatura(self, asignatura):  # **método**
        self.asignaturas.append(asignatura)

    def mostrar_asignaturas(self):
        print(f"{self.nombre} cursa: {', '.join(self.asignaturas)}")

al = Alumno("Pedro")
al.agregar_asignatura("Matemáticas")
al.agregar_asignatura("Física")
al.mostrar_asignaturas()
```

---

20. Programa para representar una cuenta corriente con límite de retiro

```
class CuentaCorriente:
    def __init__(self, titular, saldo, limite):
        self.titular = titular  # **atributo**
        self.__saldo = saldo  # **atributo privado**
        self.limite = limite  # **atributo**

    def retirar(self, monto):  # **método**
        if monto <= self.__saldo + self.limite:
            self.__saldo -= monto
        else:
            print("Excede límite de cuenta")

    def mostrar_saldo(self):
        print(f"Saldo de {self.titular}: {self.__saldo}")

cuenta = CuentaCorriente("Lucía", 1000, 500)
cuenta.retirar(1200)
cuenta.mostrar_saldo()
```

---

21. Programa para representar un libro y verificar si es largo

```
class Libro:
    def __init__(self, titulo, autor, paginas):
        self.titulo = titulo  # **atributo**
        self.autor = autor  # **atributo**
        self.paginas = paginas  # **atributo**

    def es_largo(self):  # **método**
        return self.paginas > 300

libro1 = Libro("Python Avanzado", "Juan Pérez", 350)
print(f"{libro1.titulo} es largo: {libro1.es_largo()}")
```

---

22. Programa para manejar una tienda con varios productos

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre  # **atributo**
        self.precio = precio  # **atributo**

class Tienda:
    def __init__(self):
        self.productos = []  # **atributo lista**

    def agregar_producto(self, producto):  # **método**
        self.productos.append(producto)

    def total_inventario(self):
        return sum(p.precio for p in self.productos)

tienda = Tienda()
tienda.agregar_producto(Producto("Camiseta", 20))
tienda.agregar_producto(Producto("Pantalón", 30))
print(f"Total inventario: ${tienda.total_inventario()}")
```

---

23. Programa para representar una bicicleta y mostrar velocidad

```
class Bicicleta:
    def __init__(self, marca, velocidad):
        self.marca = marca  # **atributo**
        self.velocidad = velocidad  # **atributo**

    def acelerar(self, incremento):  # **método**
        self.velocidad += incremento

    def frenar(self, decremento):  # **método**
        self.velocidad -= decremento

bike = Bicicleta("Trek", 10)
bike.acelerar(5)
bike.frenar(3)
print(f"Velocidad actual: {bike.velocidad} km/h")
```

---

24. Programa para representar un estudiante y verificar beca

```
class Estudiante:
    def __init__(self, nombre, promedio):
        self.nombre = nombre  # **atributo**
        self.promedio = promedio  # **atributo**

    def tiene_beca(self):  # **método**
        return self.promedio >= 9

est = Estudiante("Laura", 9.5)
print(f"{est.nombre} tiene beca: {est.tiene_beca()}")
```

---

25. Programa para manejar un rectángulo y verificar si es cuadrado

```
class Rectangulo:
    def __init__(self, ancho, alto):
        self.ancho = ancho  # **atributo**
        self.alto = alto    # **atributo**

    def es_cuadrado(self):  # **método**
        return self.ancho == self.alto

rect = Rectangulo(5, 5)
print(f"Es cuadrado: {rect.es_cuadrado()}")
```

---

26. Programa para representar un vehículo con método de descripción

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca  # **atributo**
        self.modelo = modelo  # **atributo**

    def descripcion(self):  # **método**
        return f"{self.marca} {self.modelo}"

v = Vehiculo("Honda", "Civic")
print(v.descripcion())
```

27. Programa para representar un banco y múltiples cuentas

```
class Cuenta:
    def __init__(self, titular, saldo):
        self.titular = titular  # **atributo**
        self.saldo = saldo  # **atributo**

class Banco:
    def __init__(self):
        self.cuentas = []  # **atributo lista**

    def agregar_cuenta(self, cuenta):  # **método**
        self.cuentas.append(cuenta)

    def total_saldo(self):
        return sum(c.saldo for c in self.cuentas)

banco = Banco()
banco.agregar_cuenta(Cuenta("Ana", 1000))
banco.agregar_cuenta(Cuenta("Luis", 500))
print(f"Total saldo en banco: ${banco.total_saldo()}")
```

---

28. Programa para representar un empleado con método de promoción

```
class Empleado:
    def __init__(self, nombre, puesto):
        self.nombre = nombre  # **atributo**
        self.puesto = puesto  # **atributo**

    def promover(self, nuevo_puesto):  # **método**
        self.puesto = nuevo_puesto

emp = Empleado("Carlos", "Junior")
emp.promover("Senior")
print(f"{emp.nombre} ahora es {emp.puesto}")
```

---

29. Programa para representar un animal y su sonido

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre  # **atributo**

    def hacer_sonido(self):  # **método genérico**
        print(f"{self.nombre} hace un sonido")

class Perro(Animal):
    def hacer_sonido(self):  # **sobrescritura**
        print(f"{self.nombre} dice Guau!")

p = Perro("Rex")
p.hacer_sonido()
```

---

30. Programa para manejar un carrito de compras con productos y total

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre  # **atributo**
        self.precio = precio  # **atributo**

class Carrito:
    def __init__(self):
        self.productos = []  # **atributo lista**

    def agregar_producto(self, producto):  # **método**
        self.productos.append(producto)

    def total(self):
        return sum(p.precio for p in self.productos)

carrito = Carrito()
carrito.agregar_producto(Producto("Libro", 15))
carrito.agregar_producto(Producto("Lapicera", 2))
print(f"Total carrito: ${carrito.total()}")
```

---

31. Programa para representar un estudiante y sus materias con notas

```
class Estudiante:
    def __init__(self, nombre):
        self.nombre = nombre  # **atributo**
        self.materias = {}  # **atributo diccionario**

    def agregar_nota(self, materia, nota):  # **método**
        self.materias[materia] = nota

    def promedio(self):
        return sum(self.materias.values()) / len(self.materias)

est = Estudiante("Ana")
est.agregar_nota("Matemáticas", 8)
est.agregar_nota("Historia", 9)
print(f"Promedio de {est.nombre}: {est.promedio()}")
```

---

32. Programa para manejar una lista de cuentas bancarias y mostrar saldos

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular  # **atributo**
        self.saldo = saldo  # **atributo**

    def mostrar_saldo(self):
        print(f"{self.titular}: ${self.saldo}")

cuentas = [
    CuentaBancaria("Luis", 1000),
    CuentaBancaria("Marta", 2000)
]

for c in cuentas:
    c.mostrar_saldo()
```

---

33. Programa para representar un rectángulo y calcular su perímetro

```
class Rectangulo:
    def __init__(self, ancho, alto):
        self.ancho = ancho  # **atributo**
        self.alto = alto  # **atributo**

    def perimetro(self):  # **método**
        return 2 * (self.ancho + self.alto)

rect = Rectangulo(4, 6)
print(f"Perímetro del rectángulo: {rect.perimetro()}")
```

---

34. Programa para manejar una librería con varios libros

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo  # **atributo**
        self.autor = autor  # **atributo**

class Libreria:
    def __init__(self):
        self.libros = []  # **atributo lista**

    def agregar_libro(self, libro):  # **método**
        self.libros.append(libro)

    def mostrar_libros(self):
        for l in self.libros:
            print(f"{l.titulo} por {l.autor}")

libreria = Libreria()
libreria.agregar_libro(Libro("Python Básico", "Juan"))
libreria.agregar_libro(Libro("POO Avanzada", "Ana"))
libreria.mostrar_libros()
```

---

35. Programa para representar un empleado y calcular salario anual

```
class Empleado:
    def __init__(self, nombre, salario_mensual):
        self.nombre = nombre  # **atributo**
        self.salario_mensual = salario_mensual  # **atributo**

    def salario_anual(self):  # **método**
        return self.salario_mensual * 12

emp = Empleado("Carlos", 2000)
print(f"Salario anual de {emp.nombre}: ${emp.salario_anual()}")
```

---

36. Programa para representar un círculo y comprobar si un punto está dentro

```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio  # **atributo**

    def contiene_punto(self, x, y):  # **método**
        return x**2 + y**2 <= self.radio**2

c = Circulo(5)
print(c.contiene_punto(3, 4))  # True
print(c.contiene_punto(6, 0))  # False
```

---

37. Programa para representar una persona con método de saludo personalizado

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # **atributo**
        self.edad = edad  # **atributo**

    def saludar(self):  # **método**
        print(f"Hola, soy {self.nombre} y tengo {self.edad} años")

p = Persona("Luis", 30)
p.saludar()
```

---

38. Programa para manejar vehículos con velocidad y dirección

```
class Vehiculo:
    def __init__(self, marca, velocidad=0):
        self.marca = marca  # **atributo**
        self.velocidad = velocidad  # **atributo**
        self.direccion = 0  # **atributo**

    def acelerar(self, incremento):  # **método**
        self.velocidad += incremento

    def girar(self, grados):  # **método**
        self.direccion += grados

v = Vehiculo("Toyota")
v.acelerar(50)
v.girar(90)
print(f"{v.marca} velocidad: {v.velocidad} km/h, dirección: {v.direccion}°")
```

---

39. Programa para representar una cuenta bancaria con método de transferencia

```
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular  # **atributo**
        self.saldo = saldo  # **atributo**

    def transferir(self, otra_cuenta, monto):  # **método**
        if monto <= self.saldo:
            self.saldo -= monto
            otra_cuenta.saldo += monto
        else:
            print("Saldo insuficiente")

cuenta1 = CuentaBancaria("Ana", 1000)
cuenta2 = CuentaBancaria("Luis", 500)
cuenta1.transferir(cuenta2, 300)
print(cuenta1.saldo, cuenta2.saldo)
```

---

40. Programa para representar un estudiante y mostrar materias aprobadas

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre  # **atributo**
        self.notas = notas  # **atributo diccionario**

    def materias_aprobadas(self):  # **método**
        return [m for m, n in self.notas.items() if n >= 6]

est = Estudiante("Marta", {"Matemáticas": 8, "Historia": 5, "Física": 7})
print(f"{est.nombre} aprobó: {est.materias_aprobadas()}")
```

---

41. Programa para representar un curso y sus estudiantes

```
class Estudiante:
    def __init__(self, nombre):
        self.nombre = nombre  # **atributo**

class Curso:
    def __init__(self, nombre):
        self.nombre = nombre  # **atributo**
        self.estudiantes = []  # **atributo lista**

    def agregar_estudiante(self, estudiante):  # **método**
        self.estudiantes.append(estudiante)

    def mostrar_estudiantes(self):
        for e in self.estudiantes:
            print(e.nombre)

curso = Curso("Python Básico")
curso.agregar_estudiante(Estudiante("Ana"))
curso.agregar_estudiante(Estudiante("Luis"))
curso.mostrar_estudiantes()
```

---

42. Programa para representar un teléfono y realizar llamadas

```
class Telefono:
    def __init__(self, marca):
        self.marca = marca  # **atributo**
        self.estado = "apagado"  # **atributo**

    def encender(self):  # **método**
        self.estado = "encendido"

    def llamar(self, numero):  # **método**
        if self.estado == "encendido":
            print(f"Llamando al {numero}")
        else:
            print("El teléfono está apagado")

t = Telefono("Samsung")
t.encender()
t.llamar("123456789")
```

---

43. Programa para manejar un zoológico con animales

```
class Animal:
    def __init__(self, nombre, especie):
        self.nombre = nombre  # **atributo**
        self.especie = especie  # **atributo**

class Zoologico:
    def __init__(self):
        self.animales = []  # **atributo lista**

    def agregar_animal(self, animal):  # **método**
        self.animales.append(animal)

    def mostrar_animales(self):
        for a in self.animales:
            print(f"{a.nombre} ({a.especie})")

zoo = Zoologico()
zoo.agregar_animal(Animal("Rex", "Perro"))
zoo.agregar_animal(Animal("Luna", "Gato"))
zoo.mostrar_animales()
```

---

44. Programa para representar un banco con cuentas y depósitos

```
class CuentaBancaria:
    def __init__(self, titular, saldo=0):
        self.titular = titular  # **atributo**
        self.saldo = saldo  # **atributo**

    def depositar(self, monto):  # **método**
        self.saldo += monto

cuentas = [
    CuentaBancaria("Ana"),
    CuentaBancaria("Luis", 500)
]

cuentas[0].depositar(300)
for c in cuentas:
    print(f"{c.titular} saldo: ${c.saldo}")
```

---

45. Programa para representar un reloj y mostrar la hora

```
class Reloj:
    def __init__(self, hora, minuto):
        self.hora = hora  # **atributo**
        self.minuto = minuto  # **atributo**

    def mostrar_hora(self):  # **método**
        print(f"{self.hora:02d}:{self.minuto:02d}")

r = Reloj(14, 30)
r.mostrar_hora()
```

---

46. Programa para representar una película y su duración

```
class Pelicula:
    def __init__(self, titulo, duracion):
        self.titulo = titulo  # **atributo**
        self.duracion = duracion  # **atributo en minutos**

    def es_larga(self):  # **método**
        return self.duracion > 120

p = Pelicula("Avengers", 150)
print(f"{p.titulo} es larga: {p.es_larga()}")
```

---

47. Programa para representar un punto 3D y calcular distancia a otro punto

```
import math

class Punto3D:
    def __init__(self, x, y, z):
        self.x = x  # **atributo**
        self.y = y  # **atributo**
        self.z = z  # **atributo**

    def distancia(self, otro):  # **método**
        return math.sqrt((self.x - otro.x)**2 + (self.y - otro.y)**2 + (self.z -
otro.z)**2)

p1 = Punto3D(0, 0, 0)
p2 = Punto3D(3, 4, 12)
print(f"Distancia: {p1.distancia(p2)}")
```

---

48. Programa para representar un avión y mostrar velocidad máxima

```
class Avion:
    def __init__(self, modelo, velocidad_max):
        self.modelo = modelo  # **atributo**
        self.velocidad_max = velocidad_max  # **atributo**

    def mostrar_info(self):  # **método**
        print(f"{self.modelo} velocidad máxima: {self.velocidad_max} km/h")

a = Avion("Boeing 747", 920)
a.mostrar_info()
```

---

49. Programa para manejar una lista de empleados y calcular salario promedio

```
class Empleado:
    def __init__(self, nombre, salario):
        self.nombre = nombre  # **atributo**
        self.salario = salario  # **atributo**

empleados = [
    Empleado("Ana", 2000),
    Empleado("Luis", 2500),
    Empleado("Carlos", 3000)
]

promedio = sum(e.salario for e in empleados) / len(empleados)
print(f"Salario promedio: ${promedio}")
```

---

50. Programa para representar un ordenador con encendido y apagado

```
class Ordenador:
    def __init__(self, marca):
        self.marca = marca  # **atributo**
        self.encendido = False  # **atributo**

    def encender(self):  # **método**
        self.encendido = True

    def apagar(self):  # **método**
        self.encendido = False

    def estado(self):
        print(f"{self.marca} está {'encendido' if self.encendido else
'apagado'}")

pc = Ordenador("Dell")
pc.encender()
pc.estado()
pc.apagar()
pc.estado()
```

---

10. LECCIÓN DE PROGRAMACIÓN EN PYTHON: ARCHIVOS Y DIRECTORIOS
Tamaño de fuente: 12

1. INTRODUCCIÓN A ARCHIVOS Y DIRECTORIOS
Python permite **crear, leer, escribir, modificar y eliminar archivos**, así como **navegar y manipular directorios**. Trabajar con archivos es fundamental para almacenar datos persistentes y organizar información en proyectos.

Existen **tipos de archivos** más comunes:

- Archivos de texto: contienen datos legibles por humanos (.txt, .csv).
- Archivos binarios: contienen datos no legibles directamente (.jpg, .pdf, .bin).

Los **directorios** (o carpetas) permiten organizar archivos jerárquicamente. Python ofrece herramientas para interactuar con ellos de manera programática.

---

2. ABRIR Y CERRAR ARCHIVOS

Para trabajar con archivos, se utiliza la función **open()**, que requiere al menos dos parámetros: la ruta del archivo y el modo de apertura.

Modos de apertura principales:

- **'r'**: lectura (archivo debe existir).
- **'w'**: escritura (crea archivo nuevo o sobrescribe existente).
- **'a'**: anexar (agrega contenido al final del archivo).
- **'x'**: creación exclusiva (falla si el archivo ya existe).
- **'b'**: modo binario (combinable, ej. 'rb', 'wb').
- **'t'**: modo texto (por defecto, ej. 'rt').

Ejemplo:
```
archivo = open('datos.txt', 'r')
contenido = archivo.read()
archivo.close()
```

Nota: Siempre cerrar archivos después de usarlos para liberar recursos.

---

3. CONTEXTO MANEJADO CON WITH

Python recomienda usar **with** para manejar archivos, ya que cierra automáticamente el archivo al finalizar el bloque:

Ejemplo:
```
with open('datos.txt', 'r') as archivo:
contenido = archivo.read()
```

No es necesario archivo.close()

---

4. MÉTODOS PARA LEER ARCHIVOS

- **read()**: lee todo el contenido como una cadena.
- **readline()**: lee línea por línea.
- **readlines()**: devuelve una lista con todas las líneas.

Ejemplo:
with open('datos.txt', 'r') as archivo:
for linea in archivo:
print(linea.strip())

Nota: strip() elimina saltos de línea al final.

---

5. ESCRIBIR Y ANEXAR ARCHIVOS

- **write()**: escribe texto en el archivo (sobrescribe si modo 'w').
- **writelines()**: escribe múltiples líneas de una lista.

Ejemplo:
with open('registro.txt', 'w') as archivo:
archivo.write('Primera línea\n')
archivo.write('Segunda línea\n')

Para anexar:
with open('registro.txt', 'a') as archivo:
archivo.write('Tercera línea añadida\n')

---

6. TRABAJAR CON RUTAS Y DIRECTORIOS

Python proporciona los módulos **os** y **pathlib** para manipular directorios y rutas.

Usando os:

- os.getcwd(): devuelve el directorio actual.
- os.chdir('ruta'): cambia el directorio actual.
- os.listdir('ruta'): lista archivos y carpetas en un directorio.
- os.mkdir('nueva_carpeta'): crea un directorio.
- os.makedirs('ruta/completa'): crea directorios anidados.
- os.rmdir('carpeta'): elimina un directorio vacío.
- os.remove('archivo.txt'): elimina un archivo.

Usando pathlib:
from pathlib import Path
ruta = Path('archivo.txt')

- ruta.exists(): verifica existencia.
- ruta.is_file(): verifica si es archivo.
- ruta.is_dir(): verifica si es directorio.

Ejemplo de uso de pathlib:
from pathlib import Path
carpeta = Path('proyecto')
if not carpeta.exists():
carpeta.mkdir()

---

7. MÉTODOS AVANZADOS DE LECTURA Y ESCRITURA

- **read(size)**: lee una cantidad específica de caracteres o bytes.
- **seek(offset, whence)**: mueve el puntero de lectura/escritura.
- **tell()**: devuelve la posición actual del puntero.

Ejemplo:
with open('datos.txt', 'r') as archivo:
archivo.seek(10)
print(archivo.read(5))

---

8. MANEJO DE ERRORES EN ARCHIVOS

Se recomienda usar **try-except** para manejar errores como archivos inexistentes:

Ejemplo:
try:
with open('archivo_inexistente.txt', 'r') as archivo:
contenido = archivo.read()
except FileNotFoundError:
print('El archivo no existe')

---

9. EJERCICIOS PRÁCTICOS SUGERIDOS

1. Crear un archivo `notas.txt` y escribir 5 líneas con nombres de estudiantes.
2. Leer el archivo y mostrar solo las líneas que contienen la letra "a".
3. Crear una carpeta `informes` y mover `notas.txt` dentro de ella.
4. Contar cuántos archivos `.txt` existen en el directorio actual.
5. Crear un archivo binario y escribir datos en formato bytes.

---

10. RESUMEN DE CONCEPTOS CLAVE

- Abrir archivos: open() y with.
- Modos: 'r', 'w', 'a', 'x', 'b', 't'.
- Métodos de lectura: read(), readline(), readlines().
- Métodos de escritura: write(), writelines().
- Directorios: os y pathlib para manipular rutas, crear y eliminar carpetas.
- Manejo de errores: try-except.
- Avanzado: seek(), tell(), manejo de archivos binarios.

---

1. Crear un archivo de texto y escribir un mensaje
Enunciado: Crear un archivo llamado **mensaje.txt** y escribir "Hola, Python".
Código:

```
with open('mensaje.txt', 'w') as archivo:
archivo.write('Hola, Python\n')
print('Archivo creado y mensaje escrito')
```

---

2. Leer el contenido de un archivo existente
Enunciado: Leer el archivo **mensaje.txt** y mostrar su contenido en pantalla.
Código:

```
with open('mensaje.txt', 'r') as archivo:
contenido = archivo.read()
print('Contenido del archivo:')
print(contenido)
```

---

3. Añadir texto a un archivo existente
Enunciado: Añadir la línea "Bienvenido al mundo de Python" al archivo **mensaje.txt**.
Código:

```
with open('mensaje.txt', 'a') as archivo:
archivo.write('Bienvenido al mundo de Python\n')
print('Texto añadido al archivo')
```

---

4. Contar líneas de un archivo
Enunciado: Contar cuántas líneas tiene el archivo **mensaje.txt**.
Código:

```
with open('mensaje.txt', 'r') as archivo:
lineas = archivo.readlines()
print('Número de líneas en el archivo:', len(lineas))
```

---

5. Leer archivo línea por línea
Enunciado: Mostrar cada línea del archivo **mensaje.txt** de manera individual.
Código:

```
with open('mensaje.txt', 'r') as archivo:
for linea in archivo:
print(linea.strip())
```

---

6. Crear múltiples archivos en un directorio
Enunciado: Crear un directorio llamado **documentos** y dentro crear tres archivos: **doc1.txt**, **doc2.txt**, **doc3.txt** con contenido diferente.
Código:

```
import os

if not os.path.exists('documentos'):
os.mkdir('documentos')

for i in range(1, 4):
with open(f'documentos/doc{i}.txt', 'w') as archivo:
archivo.write(f'Contenido del documento {i}\n')
print('Archivos creados dentro de documentos')
```

---

7. Listar archivos en un directorio
Enunciado: Mostrar todos los archivos dentro del directorio **documentos**.
Código:

```
import os

archivos = os.listdir('documentos')
print('Archivos en documentos:')
for archivo in archivos:
print(archivo)
```

---

8. Verificar si un archivo existe
Enunciado: Verificar si el archivo **doc1.txt** existe dentro de **documentos**.
Código:

```
import os

ruta = 'documentos/doc1.txt'
if os.path.exists(ruta):
print('El archivo existe')
else:
print('El archivo no existe')
```

---

9. Copiar el contenido de un archivo a otro
Enunciado: Copiar el contenido de **doc1.txt** a un nuevo archivo **copia_doc1.txt**.
Código:

```
with open('documentos/doc1.txt', 'r') as original:
contenido = original.read()

with open('documentos/copia_doc1.txt', 'w') as copia:
copia.write(contenido)

print('Archivo copiado exitosamente')
```

---

10. Contar cuántas palabras tiene un archivo
Enunciado: Contar el número de palabras en **doc1.txt**.
Código:
```
with open('documentos/doc1.txt', 'r') as archivo:
contenido = archivo.read()
palabras = contenido.split()
print('Número de palabras en doc1.txt:', len(palabras))
```

Perfecto. Aquí van los **programas del 11 al 20**, con el mismo formato: enunciado en **negrita**, partes importantes en **negrita**, tamaño de fuente 12, nivel de dificultad de principiante a medio.

---

11. Crear un archivo CSV y agregar datos
Enunciado: Crear un archivo **estudiantes.csv** con columnas **Nombre** y **Edad**, y agregar 3 registros.
Código:
```
import csv

with open('estudiantes.csv', 'w', newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre', 'Edad'])
escritor.writerow(['Ana', 20])
escritor.writerow(['Luis', 22])
escritor.writerow(['Carla', 19])

print('Archivo CSV creado con 3 registros')
```

---

12. Leer un archivo CSV y mostrar sus registros
Enunciado: Leer el archivo **estudiantes.csv** y mostrar cada fila.
Código:
```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.reader(archivo)
for fila in lector:
print(fila)
```

---

13. Crear directorios anidados
Enunciado: Crear la estructura de carpetas **proyecto/datos/imagenes**.
Código:
```
import os

os.makedirs('proyecto/datos/imagenes', exist_ok=True)
print('Directorios creados exitosamente')
```

---

14. Mover un archivo a otro directorio
Enunciado: Mover **copia_doc1.txt** a la carpeta **documentos_backup** (crear la carpeta si no existe).
Código:

```
import os
import shutil

if not os.path.exists('documentos_backup'):
os.mkdir('documentos_backup')

shutil.move('documentos/copia_doc1.txt', 'documentos_backup/copia_doc1.txt')
print('Archivo movido a documentos_backup')
```

---

15. Eliminar un archivo
Enunciado: Eliminar el archivo **doc2.txt** dentro de **documentos**.
Código:

```
import os

ruta = 'documentos/doc2.txt'
if os.path.exists(ruta):
os.remove(ruta)
print('Archivo eliminado')
else:
print('El archivo no existe')
```

---

16. Eliminar un directorio vacío
Enunciado: Eliminar la carpeta **documentos_backup** (debe estar vacía).
Código:

```
import os

if os.path.exists('documentos_backup'):
os.rmdir('documentos_backup')
print('Directorio eliminado')
else:
print('El directorio no existe')
```

---

17. Leer un archivo JSON y mostrar sus datos
Enunciado: Leer un archivo **datos.json** y mostrar el contenido (crear el archivo si no existe con datos ejemplo).
Código:

```
import json

# Crear archivo JSON si no existe
datos = {'nombre': 'Ana', 'edad': 25, 'ciudad': 'Madrid'}
with open('datos.json', 'w') as archivo:
json.dump(datos, archivo)

# Leer el archivo
with open('datos.json', 'r') as archivo:
contenido = json.load(archivo)
print('Contenido del JSON:', contenido)
```

---

18. Añadir un nuevo registro a un archivo JSON
Enunciado: Añadir {'nombre': 'Luis', 'edad': 30, 'ciudad': 'Barcelona'} a **datos.json** (convertirlo en lista de registros).
Código:

```
import json

try:
with open('datos.json', 'r') as archivo:
registros = json.load(archivo)
except json.JSONDecodeError:
registros = []

# Convertir a lista si no lo es
if not isinstance(registros, list):
registros = [registros]

registros.append({'nombre': 'Luis', 'edad': 30, 'ciudad': 'Barcelona'})

with open('datos.json', 'w') as archivo:
json.dump(registros, archivo, indent=4)

print('Nuevo registro añadido al JSON')
```

---

19. Copiar todos los archivos de un directorio a otro
Enunciado: Copiar todos los archivos de **documentos** a **documentos_backup**.
Código:
```
import os
import shutil

if not os.path.exists('documentos_backup'):
os.mkdir('documentos_backup')

for archivo in os.listdir('documentos'):
ruta_origen = os.path.join('documentos', archivo)
ruta_destino = os.path.join('documentos_backup', archivo)
shutil.copy(ruta_origen, ruta_destino)

print('Todos los archivos copiados a documentos_backup')
```

---

20. Contar archivos con extensión .txt en un directorio
Enunciado: Contar cuántos archivos con extensión **.txt** existen en la carpeta **documentos**.
Código:
```
import os

contador = 0
for archivo in os.listdir('documentos'):
if archivo.endswith('.txt'):
contador += 1

print('Número de archivos .txt en documentos:', contador)
```

---

21. Leer un archivo y mostrar solo las líneas que contienen una palabra específica
Enunciado: Mostrar todas las líneas de **doc1.txt** que contengan la palabra **"contenido"**.
Código:
```
with open('documentos/doc1.txt', 'r') as archivo:
for linea in archivo:
if 'contenido' in linea:
print(linea.strip())
```

---

22. Reemplazar una palabra en un archivo
Enunciado: Reemplazar la palabra **"documento"** por **"archivo"** en **doc1.txt**.
Código:

```
ruta = 'documentos/doc1.txt'

with open(ruta, 'r') as archivo:
contenido = archivo.read()

contenido = contenido.replace('documento', 'archivo')

with open(ruta, 'w') as archivo:
archivo.write(contenido)

print('Palabra reemplazada exitosamente')
```

---

23. Crear un archivo de registro y agregar marcas de tiempo
Enunciado: Crear un archivo **log.txt** y registrar mensajes con la fecha y hora actual.
Código:

```
from datetime import datetime

with open('log.txt', 'a') as archivo:
archivo.write(f'[{datetime.now()}] Inicio de sesión\n')
archivo.write(f'[{datetime.now()}] Tarea completada\n')

print('Registro creado con marcas de tiempo')
```

---

24. Leer un archivo de números y calcular la suma
Enunciado: Leer un archivo **numeros.txt** con un número por línea y mostrar la suma total.
Código:

```
with open('numeros.txt', 'w') as archivo:
archivo.write('10\n20\n30\n40\n50\n')

suma = 0
with open('numeros.txt', 'r') as archivo:
for linea in archivo:
suma += int(linea.strip())

print('Suma total de los números:', suma)
```

---

25. Crear un archivo de texto con contenido múltiple usando writelines()
Enunciado: Crear **frutas.txt** con una lista de frutas usando **writelines()**.
Código:

```
frutas = ['Manzana\n', 'Banana\n', 'Cereza\n', 'Naranja\n']

with open('frutas.txt', 'w') as archivo:
archivo.writelines(frutas)

print('Archivo frutas.txt creado')
```

---

26. Contar cuántas veces aparece una palabra en un archivo
Enunciado: Contar cuántas veces aparece la palabra **"Python"** en **mensaje.txt**.
Código:

```
contador = 0
with open('mensaje.txt', 'r') as archivo:
for linea in archivo:
contador += linea.count('Python')

print('La palabra "Python" aparece', contador, 'veces')
```

---

27. Crear un archivo de texto temporal y eliminarlo después
Enunciado: Crear **temp.txt**, escribir un mensaje y luego eliminarlo.
Código:

```
import os

with open('temp.txt', 'w') as archivo:
archivo.write('Este es un archivo temporal\n')

print('Archivo temporal creado')

os.remove('temp.txt')
print('Archivo temporal eliminado')
```

---

28. Ordenar las líneas de un archivo alfabéticamente
Enunciado: Leer **frutas.txt** y crear **frutas_ordenadas.txt** con las líneas ordenadas alfabéticamente.
Código:

```
with open('frutas.txt', 'r') as archivo:
lineas = archivo.readlines()

lineas.sort()

with open('frutas_ordenadas.txt', 'w') as archivo:
archivo.writelines(lineas)

print('Archivo frutas_ordenadas.txt creado con líneas ordenadas')
```

---

29. Renombrar un archivo
Enunciado: Renombrar **frutas_ordenadas.txt** a **frutas_alfabetico.txt**.
Código:

```
import os

if os.path.exists('frutas_ordenadas.txt'):
os.rename('frutas_ordenadas.txt', 'frutas_alfabetico.txt')
print('Archivo renombrado exitosamente')
else:
print('El archivo no existe')
```

---

30. Leer un archivo CSV y filtrar registros por condición
Enunciado: Leer **estudiantes.csv** y mostrar solo los estudiantes mayores de 20 años.
Código:

```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Edad']) > 20:
print(fila)
```

---

31. Crear un archivo binario y escribir datos
Enunciado: Crear un archivo **datos.bin** y escribir los números del 1 al 5 en formato binario.
Código:

```
with open('datos.bin', 'wb') as archivo:
for i in range(1, 6):
archivo.write(i.to_bytes(2, byteorder='big'))

print('Archivo binario datos.bin creado')
```

---

32. Leer datos de un archivo binario
Enunciado: Leer el archivo **datos.bin** y mostrar los números guardados.
Código:

```
with open('datos.bin', 'rb') as archivo:
while byte := archivo.read(2):
numero = int.from_bytes(byte, byteorder='big')
print(numero)
```

---

33. Crear múltiples archivos automáticamente
Enunciado: Crear 5 archivos llamados **archivo1.txt** a **archivo5.txt** con un mensaje en cada uno.
Código:

```
for i in range(1, 6):
with open(f'archivo{i}.txt', 'w') as archivo:
archivo.write(f'Este es el archivo {i}\n')

print('5 archivos creados exitosamente')
```

---

34. Combinar varios archivos de texto en uno solo
Enunciado: Combinar todos los archivos **archivo1.txt** a **archivo5.txt** en un archivo **combinado.txt**.
Código:

```
with open('combinado.txt', 'w') as combinado:
for i in range(1, 6):
with open(f'archivo{i}.txt', 'r') as archivo:
combinado.write(archivo.read())

print('Archivos combinados en combinado.txt')
```

---

35. Buscar archivos por extensión en un directorio
Enunciado: Mostrar todos los archivos con extensión **.txt** en la carpeta actual.
Código:

```
import os

archivos_txt = [f for f in os.listdir('.') if f.endswith('.txt')]
print('Archivos .txt en el directorio actual:')
for archivo in archivos_txt:
print(archivo)
```

---

36. Contar caracteres, palabras y líneas de un archivo
Enunciado: Contar caracteres, palabras y líneas de **mensaje.txt**.
Código:

```
with open('mensaje.txt', 'r') as archivo:
contenido = archivo.read()

num_caracteres = len(contenido)
num_palabras = len(contenido.split())
num_lineas = len(contenido.splitlines())

print('Caracteres:', num_caracteres)
print('Palabras:', num_palabras)
print('Líneas:', num_lineas)
```

---

37. Crear un archivo de texto con un patrón
Enunciado: Crear **numeros_patron.txt** con los números del 1 al 10, cada uno en una línea.
Código:

```
with open('numeros_patron.txt', 'w') as archivo:
for i in range(1, 11):
archivo.write(f'{i}\n')

print('Archivo numeros_patron.txt creado')
```

---

38. Leer un archivo y mostrar solo números pares
Enunciado: Leer **numeros_patron.txt** y mostrar solo los números pares.
Código:

```
with open('numeros_patron.txt', 'r') as archivo:
for linea in archivo:
numero = int(linea.strip())
if numero % 2 == 0:
print(numero)
```

---

39. Guardar y leer una lista de Python en un archivo JSON
Enunciado: Guardar la lista **[10, 20, 30, 40]** en **lista.json** y luego leerla.
Código:

```
import json

lista = [10, 20, 30, 40]
with open('lista.json', 'w') as archivo:
json.dump(lista, archivo)

with open('lista.json', 'r') as archivo:
datos = json.load(archivo)

print('Contenido de lista.json:', datos)
```

---

40. Copiar un archivo solo si existe
Enunciado: Copiar **mensaje.txt** a **backup_mensaje.txt** solo si el archivo existe.
Código:

```
import shutil
import os

if os.path.exists('mensaje.txt'):
shutil.copy('mensaje.txt', 'backup_mensaje.txt')
print('Archivo copiado a backup_mensaje.txt')
else:
print('mensaje.txt no existe')
```

---

41. Leer múltiples archivos JSON y combinarlos
Enunciado: Leer los archivos **datos1.json** y **datos2.json** y combinarlos en **todos_los_datos.json**.
Código:

```
import json
import os

# Crear archivos JSON de ejemplo
with open('datos1.json', 'w') as f:
json.dump([{'nombre':'Ana','edad':25}], f)
with open('datos2.json', 'w') as f:
json.dump([{'nombre':'Luis','edad':30}], f)

# Leer y combinar
todos = []
for archivo_json in ['datos1.json','datos2.json']:
with open(archivo_json, 'r') as f:
todos.extend(json.load(f))

with open('todos_los_datos.json', 'w') as f:
json.dump(todos, f, indent=4)

print('Archivos JSON combinados en todos_los_datos.json')
```

---

42. Contar archivos y carpetas en un directorio
Enunciado: Contar cuántos archivos y cuántas carpetas hay en el directorio actual.
Código:

```
import os

archivos = 0
carpetas = 0

for item in os.listdir('.'):
if os.path.isfile(item):
archivos += 1
elif os.path.isdir(item):
carpetas += 1

print('Número de archivos:', archivos)
print('Número de carpetas:', carpetas)
```

---

43. Crear un archivo CSV con números cuadrados
Enunciado: Crear **cuadrados.csv** con dos columnas: **Número** y **Cuadrado** para números del 1 al 10.
Código:

```
import csv

with open('cuadrados.csv', 'w', newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Número','Cuadrado'])
for i in range(1,11):
escritor.writerow([i,i**2])

print('Archivo cuadrados.csv creado')
```

---

44. Leer un archivo CSV y calcular promedio
Enunciado: Leer **cuadrados.csv** y calcular el promedio de la columna **Cuadrado**.
Código:

```
import csv

total = 0
contador = 0
with open('cuadrados.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
total += int(fila['Cuadrado'])
contador += 1

promedio = total / contador
print('Promedio de cuadrados:', promedio)
```

---

45. Crear archivos separados por primera letra
Enunciado: Leer **frutas.txt** y crear un archivo para cada letra inicial con las frutas correspondientes.
Código:

```
import os

with open('frutas.txt', 'r') as archivo:
for linea in archivo:
fruta = linea.strip()
if fruta:
letra = fruta[0].upper()
with open(f'{letra}.txt','a') as f:
f.write(fruta+'\n')

print('Archivos separados por letra inicial creados')
```

---

46. Mover todos los archivos .txt a un subdirectorio
Enunciado: Mover todos los archivos con extensión **.txt** a la carpeta **textos** (crear la carpeta si no existe).
Código:
```
import os
import shutil

if not os.path.exists('textos'):
os.mkdir('textos')

for archivo in os.listdir('.'):
if archivo.endswith('.txt'):
shutil.move(archivo, os.path.join('textos', archivo))

print('Todos los archivos .txt movidos a textos')
```

---

47. Leer un archivo y mostrar solo líneas impares
Enunciado: Leer **numeros_patron.txt** y mostrar solo las líneas impares (1,3,5...).
Código:
```
with open('numeros_patron.txt', 'r') as archivo:
lineas = archivo.readlines()
for i, linea in enumerate(lineas, start=1):
if i % 2 != 0:
print(linea.strip())
```

---

48. Crear un backup de todos los archivos JSON
Enunciado: Copiar todos los archivos **.json** a la carpeta **backup_json**.
Código:
```
import os
import shutil

if not os.path.exists('backup_json'):
os.mkdir('backup_json')

for archivo in os.listdir('.'):
if archivo.endswith('.json'):
shutil.copy(archivo, os.path.join('backup_json', archivo))

print('Backup de archivos JSON creado')
```

---

49. Leer archivos CSV y crear un CSV combinado
Enunciado: Combinar **estudiantes.csv** y **cuadrados.csv** en **combinado.csv** agregando todos los registros.
Código:

```
import csv

archivos = ['estudiantes.csv','cuadrados.csv']
with open('combinado.csv','w',newline='') as salida:
escritor = None
for archivo in archivos:
with open(archivo,'r') as f:
lector = csv.reader(f)
encabezado = next(lector)
if escritor is None:
escritor = csv.writer(salida)
escritor.writerow(encabezado)
for fila in lector:
escritor.writerow(fila)

print('Archivo combinado.csv creado')
```

---

50. Manejar errores al abrir un archivo inexistente
Enunciado: Intentar leer **inexistente.txt** y mostrar un mensaje de error si no existe.
Código:

```
try:
with open('inexistente.txt','r') as archivo:
contenido = archivo.read()
print(contenido)
except FileNotFoundError:
print('Error: El archivo inexistente.txt no existe')
```

---

11. LECCIÓN DE PROGRAMACIÓN EN PYTHON: ARCHIVOS CSV
Tamaño de fuente: 12

1. INTRODUCCIÓN A CSV
Los archivos **CSV (Comma-Separated Values)** son archivos de texto que almacenan datos en forma de tabla, separando los valores por comas u otros delimitadores. Son ampliamente usados para **intercambio de datos entre aplicaciones** y **almacenamiento de información estructurada**.

En Python se utilizan principalmente los módulos **csv** y **pandas** para manipular archivos CSV. Esta lección se enfocará en el módulo **csv** estándar.

---

2. CREAR UN ARCHIVO CSV
Para crear un archivo CSV se utiliza **csv.writer**.

Ejemplo:
```
import csv

with open('estudiantes.csv', 'w', newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre', 'Edad', 'Ciudad']) # Encabezado
escritor.writerow(['Ana', 20, 'Madrid'])
escritor.writerow(['Luis', 22, 'Barcelona'])
escritor.writerow(['Carla', 19, 'Valencia'])

print('Archivo CSV creado')
```

Notas importantes:

- **newline=''** evita líneas en blanco entre registros en Windows.
- **writerow()** escribe una fila de datos.

---

3. LEER UN ARCHIVO CSV
Para leer un CSV se utiliza **csv.reader**.

Ejemplo:
```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.reader(archivo)
for fila in lector:
print(fila)
```

Puntos importantes:

- Cada **fila** es una lista de valores.
- Se puede omitir la primera fila (encabezado) si se desea.

---

4. USAR csv.DictReader PARA MEJOR ACCESO
csv.DictReader permite acceder a los valores por nombre de columna.

Ejemplo:

```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
print(f"{fila['Nombre']} tiene {fila['Edad']} años y vive en {fila['Ciudad']}")
```

Ventaja: Evita depender del índice de la columna.

---

5. ESCRIBIR VARIAS FILAS A LA VEZ

writerows() permite escribir múltiples filas en una sola operación.

Ejemplo:

```
import csv

datos = [
['Pedro', 21, 'Sevilla'],
['Laura', 23, 'Bilbao'],
['Jorge', 20, 'Zaragoza']
]

with open('estudiantes_nuevos.csv', 'w', newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre', 'Edad', 'Ciudad'])
escritor.writerows(datos)

print('Archivo CSV con varias filas creado')
```

---

6. AÑADIR DATOS A UN CSV EXISTENTE

Para anexar filas se abre el archivo en modo **'a'** (append).

Ejemplo:

```
with open('estudiantes.csv', 'a', newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Marta', 24, 'Granada'])

print('Fila añadida al CSV existente')
```

---

7. FILTRAR REGISTROS AL LEER

Se puede filtrar información mientras se lee un CSV.

Ejemplo:

```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
mayores_20 = [fila for fila in lector if int(fila['Edad']) > 20]
```

for fila in mayores_20:
print(fila)

Explicación:

- Se usa comprensión de listas para filtrar registros.
- Se convierte **Edad** a entero para comparación.

---

8. MODIFICAR DATOS EXISTENTES EN UN CSV
No se puede modificar un CSV directamente; se debe leer, modificar en memoria y reescribir.

Ejemplo:
import csv

Leer y modificar

registros = []
with open('estudiantes.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Nombre'] == 'Ana':
fila['Edad'] = '21' # Modificar edad
registros.append(fila)

Reescribir archivo

with open('estudiantes.csv', 'w', newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo, fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('Datos modificados en CSV')

---

9. ELIMINAR REGISTROS DE UN CSV
Se sigue la misma estrategia que para modificar: leer, filtrar, reescribir.

Ejemplo:
import csv

registros_filtrados = []
with open('estudiantes.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Nombre'] != 'Luis': # Eliminar a Luis
registros_filtrados.append(fila)

with open('estudiantes.csv', 'w', newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']

```
escritor = csv.DictWriter(archivo, fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros_filtrados)

print('Registro eliminado del CSV')
```

---

10. CONTAR REGISTROS, LÍNEAS Y COLUMNAS

Ejemplo:

```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.reader(archivo)
filas = list(lector)

num_lineas = len(filas) - 1 # Excluyendo encabezado
num_columnas = len(filas[0])

print('Número de registros:', num_lineas)
print('Número de columnas:', num_columnas)
```

---

11. ORDENAR UN CSV POR COLUMNA

Ejemplo:

```
import csv

with open('estudiantes.csv', 'r') as archivo:
lector = csv.DictReader(archivo)
registros = sorted(lector, key=lambda x: int(x['Edad'])) # Ordenar por Edad

with open('estudiantes_ordenados.csv', 'w', newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo, fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('CSV ordenado por Edad creado')
```

---

12. USAR DIFERENTES DELIMITADORES

El CSV no necesita ser solo comas; se puede usar **;**, |, etc.

Ejemplo:

```
import csv

with open('estudiantes_puntoycoma.csv', 'w', newline='') as archivo:
escritor = csv.writer(archivo, delimiter=';')
escritor.writerow(['Nombre','Edad','Ciudad'])
escritor.writerow(['Ana',20,'Madrid'])
escritor.writerow(['Luis',22,'Barcelona'])

print('CSV con delimitador ";" creado')
```

---

13. GUARDAR NÚMEROS DECIMALES Y FORMATO
Ejemplo:

```
import csv

datos = [['Producto','Precio'],['Manzana',1.25],['Banana',0.85]]

with open('precios.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerows(datos)

print('CSV con precios guardado')
```

---

14. LEER CSV Y CALCULAR PROMEDIOS
Ejemplo:

```
import csv

total = 0
contador = 0
with open('precios.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
total += float(fila['Precio'])
contador += 1

promedio = total / contador
print('Precio promedio:', promedio)
```

---

15. CREAR UN CSV A PARTIR DE UNA LISTA DE DICCIONARIOS
Ejemplo:

```
import csv

estudiantes = [
{'Nombre':'Ana','Edad':20,'Ciudad':'Madrid'},
{'Nombre':'Luis','Edad':22,'Ciudad':'Barcelona'}
]

with open('estudiantes_dic.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(estudiantes)

print('CSV creado a partir de lista de diccionarios')
```

---

1. Crear un archivo CSV con encabezado y un registro
Enunciado: Crear un archivo **estudiantes.csv** con columnas **Nombre**, **Edad**, **Ciudad** y agregar un registro.
Código:

```
import csv

with open('estudiantes.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre','Edad','Ciudad'])
escritor.writerow(['Ana',20,'Madrid'])

print('Archivo CSV creado con un registro')
```

---

2. Leer un archivo CSV y mostrar todas las filas
Enunciado: Leer **estudiantes.csv** y mostrar cada fila.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.reader(archivo)
for fila in lector:
print(fila)
```

---

3. Añadir un nuevo registro a un CSV existente
Enunciado: Añadir a **estudiantes.csv** el registro **Luis,22,Barcelona**.
Código:

```
import csv

with open('estudiantes.csv','a',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Luis',22,'Barcelona'])

print('Registro añadido')
```

---

4. Crear un CSV con múltiples registros usando writerows()
Enunciado: Crear **estudiantes_nuevos.csv** con 3 registros.
Código:

```
import csv

datos = [['Carla',19,'Valencia'],['Pedro',21,'Sevilla'],['Laura',23,'Bilbao']]

with open('estudiantes_nuevos.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre','Edad','Ciudad'])
escritor.writerows(datos)

print('CSV con varios registros creado')
```

---

5. Leer un CSV usando DictReader
Enunciado: Leer **estudiantes.csv** y mostrar los datos con formato.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
print(f"{fila['Nombre']} tiene {fila['Edad']} años y vive en {fila['Ciudad']}")
```

---

6. Filtrar registros por condición
Enunciado: Mostrar solo los estudiantes mayores de 20 años de **estudiantes.csv**.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Edad']) > 20:
print(fila)
```

---

7. Modificar un registro existente en CSV
Enunciado: Cambiar la edad de **Ana** a 21 en **estudiantes.csv**.
Código:

```
import csv

registros = []
with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Nombre'] == 'Ana':
fila['Edad'] = '21'
registros.append(fila)

with open('estudiantes.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('Registro modificado')
```

---

8. Eliminar un registro de un CSV
Enunciado: Eliminar a **Luis** de **estudiantes.csv**.
Código:

```
import csv

registros_filtrados = []
with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Nombre'] != 'Luis':
registros_filtrados.append(fila)

with open('estudiantes.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros_filtrados)

print('Registro eliminado')
```

---

9. Contar registros y columnas de un CSV
Enunciado: Contar cuántos registros y columnas tiene **estudiantes.csv**.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.reader(archivo)
filas = list(lector)

num_columnas = len(filas[0])
num_registros = len(filas) - 1 # Excluyendo encabezado

print('Número de columnas:', num_columnas)
print('Número de registros:', num_registros)
```

---

10. Ordenar CSV por una columna
Enunciado: Ordenar **estudiantes.csv** por **Edad** y guardar en **estudiantes_ordenados.csv**.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
registros = sorted(lector,key=lambda x: int(x['Edad']))

with open('estudiantes_ordenados.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('CSV ordenado por Edad creado')
```

---

11. Crear un CSV con diferentes delimitadores
Enunciado: Crear **estudiantes_puntoycoma.csv** usando **;** como delimitador.
Código:

```
import csv

with open('estudiantes_puntoycoma.csv','w',newline='') as archivo:
escritor = csv.writer(archivo, delimiter=';')
escritor.writerow(['Nombre','Edad','Ciudad'])
escritor.writerow(['Ana',20,'Madrid'])
escritor.writerow(['Luis',22,'Barcelona'])

print('CSV con delimitador ";" creado')
```

---

12. Leer CSV y convertir datos a lista de diccionarios
Enunciado: Leer **estudiantes.csv** y almacenar cada fila como diccionario en una lista.
Código:

```
import csv

lista_estudiantes = []
with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
lista_estudiantes.append(fila)

print('Lista de diccionarios:', lista_estudiantes)
```

---

13. Guardar una lista de diccionarios en un CSV
Enunciado: Crear **estudiantes_dic.csv** a partir de una lista de diccionarios.
Código:

```
import csv

estudiantes = [
{'Nombre':'Pedro','Edad':21,'Ciudad':'Sevilla'},
{'Nombre':'Laura','Edad':23,'Ciudad':'Bilbao'}
]

with open('estudiantes_dic.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(estudiantes)

print('CSV creado desde lista de diccionarios')
```

---

14. Calcular promedio de una columna numérica en CSV
Enunciado: Calcular el promedio de **Edad** en **estudiantes.csv**.
Código:

```
import csv

total = 0
contador = 0
with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
total += int(fila['Edad'])
contador += 1

promedio = total / contador
print('Edad promedio:', promedio)
```

---

15. Filtrar registros y guardar en un nuevo CSV
Enunciado: Guardar en **mayores_20.csv** solo los estudiantes con **Edad > 20**.
Código:

```
import csv

registros_filtrados = []
with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Edad']) > 20:
registros_filtrados.append(fila)

with open('mayores_20.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros_filtrados)

print('CSV con mayores de 20 creado')
```

---

16. Leer CSV y mostrar solo valores de una columna
Enunciado: Mostrar todos los nombres de **estudiantes.csv**.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
print(fila['Nombre'])
```

---

17. Reemplazar un valor en un CSV
Enunciado: Cambiar la ciudad de **Pedro** a **Madrid** en **estudiantes_dic.csv**.
Código:

```
import csv

registros = []
with open('estudiantes_dic.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Nombre'] == 'Pedro':
fila['Ciudad'] = 'Madrid'
registros.append(fila)

with open('estudiantes_dic.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('Ciudad actualizada en CSV')
```

---

18. Contar cuántos estudiantes hay por ciudad
Enunciado: Contar cuántos estudiantes viven en cada ciudad de **estudiantes.csv**.
Código:

```
import csv
from collections import defaultdict

conteo = defaultdict(int)
with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
conteo[fila['Ciudad']] += 1

for ciudad, cantidad in conteo.items():
print(f'{ciudad}: {cantidad}')
```

---

19. Leer CSV y mostrar registros ordenados por nombre
Enunciado: Mostrar **estudiantes.csv** ordenado alfabéticamente por **Nombre**.
Código:

```
import csv

with open('estudiantes.csv','r') as archivo:
lector = csv.DictReader(archivo)
registros = sorted(lector,key=lambda x: x['Nombre'])

for fila in registros:
print(fila)
```

---

20. Combinar dos CSV en uno
Enunciado: Combinar **estudiantes.csv** y **estudiantes_nuevos.csv** en **estudiantes_combinados.csv**.
Código:

```
import csv

archivos = ['estudiantes.csv','estudiantes_nuevos.csv']
registros_combinados = []

for archivo in archivos:
with open(archivo,'r') as f:
lector = csv.DictReader(f)
for fila in lector:
registros_combinados.append(fila)

with open('estudiantes_combinados.csv','w',newline='') as archivo:
campos = ['Nombre','Edad','Ciudad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros_combinados)

print('CSV combinado creado')
```

---

21. Crear CSV con precios y productos
Enunciado: Crear **productos.csv** con columnas **Producto** y **Precio** y agregar 3 productos.
Código:

```
import csv

with open('productos.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Producto','Precio'])
escritor.writerow(['Manzana',1.25])
escritor.writerow(['Banana',0.85])
escritor.writerow(['Naranja',1.50])

print('CSV de productos creado')
```

---

22. Leer CSV y calcular precio promedio
Enunciado: Leer **productos.csv** y calcular el promedio de **Precio**.
Código:

```
import csv

total = 0
contador = 0
with open('productos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
total += float(fila['Precio'])
contador += 1

promedio = total / contador
print('Precio promedio:', promedio)
```

---

23. Filtrar productos caros y guardar en nuevo CSV
Enunciado: Guardar en **productos_caros.csv** los productos con **Precio > 1**.
Código:

```
import csv

filtrados = []
with open('productos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if float(fila['Precio']) > 1:
filtrados.append(fila)

with open('productos_caros.csv','w',newline='') as archivo:
campos = ['Producto','Precio']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(filtrados)

print('CSV de productos caros creado')
```

---

24. Leer CSV y mostrar solo nombres de productos
Enunciado: Mostrar todos los nombres de **productos.csv**.
Código:

```
import csv

with open('productos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
print(fila['Producto'])
```

---

25. Actualizar precio de un producto
Enunciado: Cambiar el precio de **Banana** a 0.95 en **productos.csv**.
Código:

```
import csv

registros = []
with open('productos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Producto'] == 'Banana':
fila['Precio'] = '0.95'
registros.append(fila)

with open('productos.csv','w',newline='') as archivo:
campos = ['Producto','Precio']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('Precio actualizado')
```

---

26. Contar cuántos productos hay
Enunciado: Contar el número total de productos en **productos.csv**.
Código:

```
import csv

contador = 0
with open('productos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
contador += 1

print('Número de productos:', contador)
```

---

27. Ordenar productos por precio descendente
Enunciado: Crear **productos_ordenados.csv** ordenando por **Precio** de mayor a menor.
Código:

```
import csv

with open('productos.csv','r') as archivo:
lector = csv.DictReader(archivo)
registros = sorted(lector,key=lambda x: float(x['Precio']), reverse=True)

with open('productos_ordenados.csv','w',newline='') as archivo:
campos = ['Producto','Precio']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('CSV de productos ordenado creado')
```

---

28. Crear CSV de estudiantes con calificaciones
Enunciado: Crear **calificaciones.csv** con columnas **Nombre**, **Matematicas**, **Fisica**, **Quimica** y agregar 3 registros.
Código:

```
import csv

with open('calificaciones.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre','Matematicas','Fisica','Quimica'])
escritor.writerow(['Ana',85,90,88])
escritor.writerow(['Luis',78,82,80])
escritor.writerow(['Carla',92,95,89])

print('CSV de calificaciones creado')
```

---

29. Calcular promedio por estudiante
Enunciado: Leer **calificaciones.csv** y mostrar el promedio de cada estudiante.
Código:

```
import csv

with open('calificaciones.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
promedio = (int(fila['Matematicas']) + int(fila['Fisica']) + int(fila['Quimica'])) / 3
print(f"{fila['Nombre']} tiene promedio {promedio:.2f}")
```

---

30. Filtrar estudiantes aprobados en Matemáticas
Enunciado: Guardar en **aprobados_matematicas.csv** los estudiantes con **Matematicas >= 80**.
Código:

```
import csv

aprobados = []
with open('calificaciones.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Matematicas']) >= 80:
aprobados.append(fila)

with open('aprobados_matematicas.csv','w',newline='') as archivo:
campos = ['Nombre','Matematicas','Fisica','Quimica']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(aprobados)

print('CSV de aprobados en Matemáticas creado')
```

---

31. Calcular promedio de cada materia
Enunciado: Leer **calificaciones.csv** y mostrar el promedio de **Matematicas**, **Fisica** y **Quimica**.
Código:

```
import csv

suma_m = suma_f = suma_q = 0
contador = 0
with open('calificaciones.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
suma_m += int(fila['Matematicas'])
suma_f += int(fila['Fisica'])
suma_q += int(fila['Quimica'])
contador += 1

print('Promedio Matematicas:', suma_m/contador)
print('Promedio Fisica:', suma_f/contador)
print('Promedio Quimica:', suma_q/contador)
```

---

32. Crear CSV con ventas por mes
Enunciado: Crear **ventas.csv** con columnas **Mes**, **Producto**, **Cantidad** y agregar 4 registros.
Código:

```
import csv

with open('ventas.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Mes','Producto','Cantidad'])
escritor.writerow(['Enero','Manzana',50])
escritor.writerow(['Enero','Banana',30])
escritor.writerow(['Febrero','Manzana',40])
escritor.writerow(['Febrero','Banana',25])

print('CSV de ventas creado')
```

---

33. Sumar ventas por producto
Enunciado: Leer **ventas.csv** y mostrar la cantidad total vendida por producto.
Código:

```
import csv
from collections import defaultdict

ventas_totales = defaultdict(int)
with open('ventas.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
ventas_totales[fila['Producto']] += int(fila['Cantidad'])

for producto, total in ventas_totales.items():
print(f'{producto}: {total}')
```

---

34. Filtrar ventas de un mes específico
Enunciado: Guardar en **ventas_enero.csv** solo las ventas de **Enero**.
Código:

```
import csv

ventas_enero = []
with open('ventas.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Mes'] == 'Enero':
ventas_enero.append(fila)

with open('ventas_enero.csv','w',newline='') as archivo:
campos = ['Mes','Producto','Cantidad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(ventas_enero)

print('CSV de ventas de Enero creado')
```

35. Leer CSV y mostrar productos con ventas mayores a un valor
Enunciado: Mostrar productos con **Cantidad > 40** de **ventas.csv**.
Código:

```
import csv

with open('ventas.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Cantidad']) > 40:
print(fila)
```

---

36. Modificar cantidad de un registro
Enunciado: Cambiar la cantidad de **Banana** en **Febrero** a 35 en **ventas.csv**.
Código:

```
import csv

registros = []
with open('ventas.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Producto'] == 'Banana' and fila['Mes'] == 'Febrero':
fila['Cantidad'] = '35'
registros.append(fila)

with open('ventas.csv','w',newline='') as archivo:
campos = ['Mes','Producto','Cantidad']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('Cantidad actualizada')
```

---

37. Crear CSV con inventario
Enunciado: Crear **inventario.csv** con columnas **Producto**, **Stock**, **Precio** y agregar 3 productos.
Código:

```
import csv

with open('inventario.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Producto','Stock','Precio'])
escritor.writerow(['Manzana',100,1.25])
escritor.writerow(['Banana',150,0.85])
escritor.writerow(['Naranja',120,1.50])

print('CSV de inventario creado')
```

---

38. Calcular valor total de inventario
Enunciado: Leer **inventario.csv** y calcular **Stock * Precio** por producto.
Código:

```
import csv

with open('inventario.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
total = int(fila['Stock']) * float(fila['Precio'])
print(f"{fila['Producto']} - Valor total: {total}")
```

---

39. Filtrar inventario con stock bajo
Enunciado: Guardar en **inventario_bajo.csv** los productos con **Stock < 130**.
Código:

```
import csv

bajo_stock = []
with open('inventario.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Stock']) < 130:
bajo_stock.append(fila)

with open('inventario_bajo.csv','w',newline='') as archivo:
campos = ['Producto','Stock','Precio']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(bajo_stock)

print('CSV de inventario bajo creado')
```

---

40. Combinar varios CSV en uno solo
Enunciado: Combinar **ventas.csv** y **inventario.csv** en **combinado.csv**.
Código:
```
import csv

archivos = ['ventas.csv','inventario.csv']
registros_combinados = []

for archivo in archivos:
with open(archivo,'r') as f:
lector = csv.DictReader(f)
for fila in lector:
registros_combinados.append(fila)
```

Determinar campos combinados

```
campos = set()
for fila in registros_combinados:
campos.update(fila.keys())
campos = list(campos)

with open('combinado.csv','w',newline='') as archivo:
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros_combinados)

print('CSV combinado creado')
```

---

41. Leer CSV y manejar errores si no existe
Enunciado: Intentar leer **inexistente.csv** y mostrar un mensaje si el archivo no existe.
Código:
```
import csv

try:
with open('inexistente.csv','r') as archivo:
lector = csv.reader(archivo)
for fila in lector:
print(fila)
except FileNotFoundError:
print('Error: El archivo inexistente.csv no existe')
```

---

42. Crear CSV grande automáticamente
Enunciado: Crear **numeros.csv** con columnas **Número** y **Cuadrado** para números del 1 al 100.
Código:

```
import csv

with open('numeros.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Número','Cuadrado'])
for i in range(1,101):
escritor.writerow([i,i**2])

print('CSV numeros.csv creado con 100 registros')
```

---

43. Leer CSV grande y calcular suma total de cuadrados
Enunciado: Leer **numeros.csv** y calcular la suma de la columna **Cuadrado**.
Código:

```
import csv

suma = 0
with open('numeros.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
suma += int(fila['Cuadrado'])

print('Suma de cuadrados:', suma)
```

---

44. Filtrar números pares y guardarlos en nuevo CSV
Enunciado: Guardar en **numeros_pares.csv** solo los números pares de **numeros.csv**.
Código:

```
import csv

pares = []
with open('numeros.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Número']) % 2 == 0:
pares.append(fila)

with open('numeros_pares.csv','w',newline='') as archivo:
campos = ['Número','Cuadrado']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(pares)

print('CSV numeros_pares.csv creado')
```

---

45. Crear CSV con alumnos y notas aleatorias
Enunciado: Crear **alumnos.csv** con columnas **Nombre**, **Nota** y generar 10 registros aleatorios.
Código:

```
import csv
import random

nombres = ['Ana','Luis','Carla','Pedro','Laura','Jorge','Marta','David','Sofia','Alberto']

with open('alumnos.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre','Nota'])
for nombre in nombres:
escritor.writerow([nombre, random.randint(50,100)])

print('CSV alumnos.csv creado con notas aleatorias')
```

---

46. Leer CSV y mostrar alumnos aprobados
Enunciado: Mostrar los alumnos de **alumnos.csv** con **Nota >= 70**.
Código:

```
import csv

with open('alumnos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Nota']) >= 70:
print(fila)
```

---

47. Actualizar nota de un alumno específico
Enunciado: Cambiar la nota de **Pedro** a 95 en **alumnos.csv**.
Código:

```
import csv

registros = []
with open('alumnos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if fila['Nombre'] == 'Pedro':
fila['Nota'] = '95'
registros.append(fila)

with open('alumnos.csv','w',newline='') as archivo:
campos = ['Nombre','Nota']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(registros)

print('Nota de Pedro actualizada')
```

---

48. Calcular promedio de notas de todos los alumnos
Enunciado: Leer **alumnos.csv** y calcular el promedio de **Nota**.
Código:

```
import csv

total = 0
contador = 0
with open('alumnos.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
total += int(fila['Nota'])
contador += 1

promedio = total / contador
print('Promedio de notas:', promedio)
```

---

49. Crear CSV con empleados y su salario
Enunciado: Crear **empleados.csv** con columnas **Nombre**, **Departamento**, **Salario** y agregar 5 registros.
Código:

```
import csv

empleados = [
['Ana','Ventas',1500],
['Luis','Marketing',1800],
['Carla','Finanzas',2000],
['Pedro','IT',2200],
['Laura','RRHH',1700]
]

with open('empleados.csv','w',newline='') as archivo:
escritor = csv.writer(archivo)
escritor.writerow(['Nombre','Departamento','Salario'])
escritor.writerows(empleados)

print('CSV empleados.csv creado')
```

---

50. Filtrar empleados con salario mayor a 1800 y guardar
Enunciado: Guardar en **empleados_altos.csv** los empleados con **Salario > 1800**.
Código:

```
import csv

altos_salario = []
with open('empleados.csv','r') as archivo:
lector = csv.DictReader(archivo)
for fila in lector:
if int(fila['Salario']) > 1800:
altos_salario.append(fila)

with open('empleados_altos.csv','w',newline='') as archivo:
campos = ['Nombre','Departamento','Salario']
escritor = csv.DictWriter(archivo,fieldnames=campos)
escritor.writeheader()
escritor.writerows(altos_salario)

print('CSV empleados con salario alto creado')
```

---

12. LECCIÓN: PROGRAMACIÓN EN PYTHON – ARCHIVOS JSON

Objetivo de la lección:
Aprender a manejar archivos JSON en Python, comprendiendo su estructura, cómo leerlos, escribirlos, modificarlos y buenas prácticas en su uso.

---

1. Introducción a JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos que es fácil de leer y escribir para los humanos y fácil de interpretar para las máquinas.

Características principales:

- Es texto plano.
- Se basa en pares **clave-valor**.
- Permite almacenar datos estructurados, como listas y diccionarios.

Ejemplo de JSON:

```
{
  "nombre": "Juan",
  "edad": 30,
  "ciudad": "Madrid",
  "hobbies": ["leer", "futbol", "programación"]
}
```

---

2. El módulo `json` en Python

Python incluye un módulo estándar llamado **`json`** que permite trabajar con JSON de manera sencilla.

Funciones principales:

1. **`json.dump(obj, file)`** – Escribe un objeto Python en un archivo en formato JSON.
2. **`json.dumps(obj)`** – Convierte un objeto Python a una cadena JSON.
3. **`json.load(file)`** – Lee un archivo JSON y lo convierte en un objeto Python.
4. **`json.loads(string)`** – Convierte una cadena JSON a un objeto Python.

Nota: Los objetos Python compatibles son principalmente:

- Diccionarios → JSON Object
- Listas → JSON Array
- Strings → JSON String
- Números → JSON Number

- Booleanos → JSON Boolean
- `None` → JSON null

---

3. Escribir JSON en un archivo

Ejemplo práctico:

```
import json

# Datos a almacenar
persona = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Barcelona",
    "hobbies": ["pintura", "correr", "programación"]
}

# Crear un archivo JSON y escribir los datos
with open("persona.json", "w") as archivo:
    json.dump(persona, archivo, indent=4)
```

Explicación:

- `with open("persona.json", "w") as archivo:` → abre un archivo en modo escritura.
- `json.dump(persona, archivo, indent=4)` → convierte el diccionario `persona` en JSON y lo escribe en el archivo.
- `indent=4` → organiza el JSON con sangrías para que sea legible.

---

4. Leer JSON desde un archivo

Ejemplo práctico:

```
import json

# Abrir y leer archivo JSON
with open("persona.json", "r") as archivo:
    datos = json.load(archivo)

print(datos)
print(datos["nombre"])
print(datos["hobbies"][1])
```

Explicación:

- `json.load(archivo)` convierte el contenido JSON en un diccionario de Python.
- Se puede acceder a los valores como en cualquier diccionario.

---

5. Convertir objetos Python a JSON (string)

Ejemplo:

```
import json

datos = {"producto": "Laptop", "precio": 1200, "disponible": True}

# Convertir a cadena JSON
json_string = json.dumps(datos)
print(json_string)
```

Opciones útiles en `dumps`:

- `indent=4` → formato legible
- `sort_keys=True` → ordena las claves alfabéticamente
- `ensure_ascii=False` → permite caracteres especiales (acentos, ñ, etc.)

---

6. Convertir JSON (string) a objeto Python

Ejemplo:

```
import json

json_string = '{"producto": "Laptop", "precio": 1200,
"disponible": true}'

datos = json.loads(json_string)
print(datos)
print(type(datos))  # <class 'dict'>
```

---

7. Modificar datos en un archivo JSON

Ejemplo práctico:

```
import json

# Leer el archivo
with open("persona.json", "r") as archivo:
    datos = json.load(archivo)

# Modificar datos
datos["edad"] = 26
datos["hobbies"].append("viajar")

# Guardar cambios
```

```
with open("persona.json", "w") as archivo:
    json.dump(datos, archivo, indent=4)
```

---

8. Trabajar con listas de objetos JSON

Ejemplo:

```
import json

personas = [
    {"nombre": "Luis", "edad": 28},
    {"nombre": "Marta", "edad": 32}
]

# Guardar lista en JSON
with open("personas.json", "w") as archivo:
    json.dump(personas, archivo, indent=4)

# Leer lista desde JSON
with open("personas.json", "r") as archivo:
    datos = json.load(archivo)

print(datos[0]["nombre"])  # Luis
```

---

9. Buenas prácticas al trabajar con JSON en Python

1. **Usar `with open()`** para manejar archivos y evitar errores de cierre.
2. **Validar JSON** antes de procesarlo si proviene de fuentes externas.
3. **Usar `indent`** para mantener archivos legibles.
4. **Evitar modificar directamente archivos grandes**, mejor leer, modificar en memoria y escribir de nuevo.
5. **Usar `ensure_ascii=False`** si se manejan caracteres especiales.

---

10. Resumen de comandos importantes

Función	Descripción
`json.dump(obj, file, indent=4)`	Escribe un objeto Python en un archivo JSON
`json.dumps(obj, indent=4)`	Convierte un objeto Python a una cadena JSON
`json.load(file)`	Lee un archivo JSON y devuelve un objeto Python
`json.loads(string)`	Convierte una cadena JSON a un objeto Python

---

1. Guardar información de una persona en un archivo JSON

Enunciado: Crear un programa que almacene el nombre, edad y ciudad de una persona en un archivo JSON.

```
import json

persona = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}

with open("persona.json", "w") as archivo:
    json.dump(persona, archivo, indent=4)

print("Datos guardados correctamente")
```

---

2. Leer información de un archivo JSON

Enunciado: Leer los datos de la persona almacenada en `persona.json` y mostrarlos.

```
import json

with open("persona.json", "r") as archivo:
    persona = json.load(archivo)

print("Nombre:", persona["nombre"])
print("Edad:", persona["edad"])
print("Ciudad:", persona["ciudad"])
```

---

3. Agregar un hobby a la información de la persona

Enunciado: Modificar la persona agregando un nuevo hobby.

```
import json

with open("persona.json", "r") as archivo:
    persona = json.load(archivo)

if "hobbies" not in persona:
    persona["hobbies"] = []

persona["hobbies"].append("leer")

with open("persona.json", "w") as archivo:
    json.dump(persona, archivo, indent=4)

print("Hobby agregado")
```

---

4. Guardar una lista de personas en un archivo JSON

Enunciado: Crear un programa que guarde varias personas en un archivo JSON.

```
import json

personas = [
    {"nombre": "Ana", "edad": 25},
    {"nombre": "Luis", "edad": 28},
    {"nombre": "Marta", "edad": 32}
]

with open("personas.json", "w") as archivo:
    json.dump(personas, archivo, indent=4)

print("Lista de personas guardada")
```

---

5. Leer y mostrar todas las personas de un archivo JSON

Enunciado: Mostrar los nombres y edades de todas las personas en `personas.json`.

```
import json

with open("personas.json", "r") as archivo:
    personas = json.load(archivo)

for persona in personas:
    print("Nombre:", persona["nombre"], "- Edad:", persona["edad"])
```

---

6. Buscar una persona por nombre

Enunciado: Pedir al usuario un nombre y mostrar los datos de esa persona si existe en `personas.json`.

```
import json

nombre_buscar = input("Ingrese el nombre de la persona: ")

with open("personas.json", "r") as archivo:
    personas = json.load(archivo)

encontrado = False
for persona in personas:
    if persona["nombre"].lower() == nombre_buscar.lower():
        print(persona)
        encontrado = True

if not encontrado:
    print("Persona no encontrada")
```

---

7. Eliminar una persona de la lista

Enunciado: Pedir un nombre al usuario y eliminar esa persona de `personas.json`.

```
import json


nombre_eliminar = input("Ingrese el nombre de la persona a
eliminar: ")


with open("personas.json", "r") as archivo:
    personas = json.load(archivo)


personas = [p for p in personas if p["nombre"].lower() !=
nombre_eliminar.lower()]


with open("personas.json", "w") as archivo:
    json.dump(personas, archivo, indent=4)


print("Persona eliminada si existía")
```

---

8. Actualizar la edad de una persona

Enunciado: Cambiar la edad de una persona según su nombre.

```
import json

nombre = input("Ingrese el nombre de la persona: ")
nueva_edad = int(input("Ingrese la nueva edad: "))

with open("personas.json", "r") as archivo:
    personas = json.load(archivo)

for persona in personas:
    if persona["nombre"].lower() == nombre.lower():
        persona["edad"] = nueva_edad

with open("personas.json", "w") as archivo:
    json.dump(personas, archivo, indent=4)

print("Edad actualizada")
```

---

9. Guardar productos en un archivo JSON

Enunciado: Crear un programa que guarde productos con nombre, precio y cantidad.

```
import json

productos = [
    {"nombre": "Laptop", "precio": 1200, "cantidad": 5},
    {"nombre": "Mouse", "precio": 25, "cantidad": 50}
]

with open("productos.json", "w") as archivo:
    json.dump(productos, archivo, indent=4)

print("Productos guardados")
```

10. Calcular el total de productos en inventario

Enunciado: Sumar la cantidad de todos los productos en `productos.json`.

```
import json

with open("productos.json", "r") as archivo:
    productos = json.load(archivo)

total = sum(p["cantidad"] for p in productos)
print("Cantidad total de productos:", total)
```

---

11. Buscar un producto por nombre

Enunciado: Pedir un nombre de producto y mostrar sus detalles si existe.

```
import json

nombre = input("Ingrese el nombre del producto: ")

with open("productos.json", "r") as archivo:
    productos = json.load(archivo)

encontrado = False
for p in productos:
    if p["nombre"].lower() == nombre.lower():
        print(p)
        encontrado = True

if not encontrado:
    print("Producto no encontrado")
```

---

12. Actualizar precio de un producto

Enunciado: Cambiar el precio de un producto según su nombre.

```
import json

nombre = input("Ingrese el nombre del producto: ")
nuevo_precio = float(input("Ingrese el nuevo precio: "))

with open("productos.json", "r") as archivo:
    productos = json.load(archivo)

for p in productos:
    if p["nombre"].lower() == nombre.lower():
        p["precio"] = nuevo_precio

with open("productos.json", "w") as archivo:
    json.dump(productos, archivo, indent=4)

print("Precio actualizado")
```

---

13. Guardar notas de estudiantes

Enunciado: Crear un archivo JSON con nombre de estudiante y calificaciones.

```
import json

estudiantes = [
    {"nombre": "Carlos", "notas": [8, 9, 7]},
    {"nombre": "Laura", "notas": [10, 9, 10]}
]

with open("estudiantes.json", "w") as archivo:
    json.dump(estudiantes, archivo, indent=4)

print("Notas guardadas")
```

14. Calcular promedio de cada estudiante

Enunciado: Leer `estudiantes.json` y mostrar el promedio de cada uno.

```
import json

with open("estudiantes.json", "r") as archivo:
    estudiantes = json.load(archivo)

for estudiante in estudiantes:
    promedio = sum(estudiante["notas"]) / len(estudiante["notas"])
    print(estudiante["nombre"], "- Promedio:", round(promedio, 2))
```

---

15. Agregar una nueva calificación a un estudiante

Enunciado: Pedir nombre y calificación y agregarla al estudiante.

```
import json

nombre = input("Ingrese el nombre del estudiante: ")
nota = float(input("Ingrese la nueva nota: "))

with open("estudiantes.json", "r") as archivo:
    estudiantes = json.load(archivo)

for estudiante in estudiantes:
    if estudiante["nombre"].lower() == nombre.lower():
        estudiante["notas"].append(nota)

with open("estudiantes.json", "w") as archivo:
    json.dump(estudiantes, archivo, indent=4)

print("Nota agregada")
```

Perfecto, continuamos con los siguientes **programas 16 a 30** de aplicaciones prácticas de Python con JSON. Mantengo la misma estructura: **enunciados numerados en negrita** y partes importantes en **negrita**.

---

16. Guardar una lista de tareas pendientes

Enunciado: Crear un archivo JSON para almacenar tareas con descripción y estado (pendiente/completada).

```
import json

tareas = [
    {"descripcion": "Hacer la compra", "estado": "pendiente"},
    {"descripcion": "Lavar ropa", "estado": "pendiente"}
]

with open("tareas.json", "w") as archivo:
    json.dump(tareas, archivo, indent=4)

print("Tareas guardadas")
```

---

17. Mostrar todas las tareas

Enunciado: Leer el archivo `tareas.json` y mostrar las tareas con su estado.

```
import json

with open("tareas.json", "r") as archivo:
    tareas = json.load(archivo)

for i, tarea in enumerate(tareas, start=1):
    print(f"{i}. {tarea['descripcion']} - Estado:
{tarea['estado']}")
```

---

18. Marcar una tarea como completada

Enunciado: Cambiar el estado de una tarea a "completada" según el índice que ingrese el usuario.

```
import json

indice = int(input("Ingrese el número de la tarea a completar: "))
- 1

with open("tareas.json", "r") as archivo:
    tareas = json.load(archivo)

if 0 <= indice < len(tareas):
    tareas[indice]["estado"] = "completada"

with open("tareas.json", "w") as archivo:
    json.dump(tareas, archivo, indent=4)

print("Tarea actualizada")
```

---

19. Eliminar una tarea

Enunciado: Eliminar una tarea según el número ingresado por el usuario.

```
import json

indice = int(input("Ingrese el número de la tarea a eliminar: ")) - 1

with open("tareas.json", "r") as archivo:
    tareas = json.load(archivo)

if 0 <= indice < len(tareas):
    del tareas[indice]

with open("tareas.json", "w") as archivo:
    json.dump(tareas, archivo, indent=4)

print("Tarea eliminada")
```

---

20. Guardar datos de clientes

Enunciado: Crear un archivo JSON con información de clientes: nombre, email y teléfono.

```
import json

clientes = [
    {"nombre": "Pedro", "email": "pedro@mail.com", "telefono": "123456789"},
    {"nombre": "Lucia", "email": "lucia@mail.com", "telefono": "987654321"}
]

with open("clientes.json", "w") as archivo:
    json.dump(clientes, archivo, indent=4)

print("Clientes guardados")
```

21. Buscar un cliente por email

Enunciado: Pedir un email y mostrar los datos del cliente si existe.

```
import json

email = input("Ingrese el email del cliente: ")

with open("clientes.json", "r") as archivo:
    clientes = json.load(archivo)

encontrado = False
for cliente in clientes:
    if cliente["email"].lower() == email.lower():
        print(cliente)
        encontrado = True

if not encontrado:
    print("Cliente no encontrado")
```

---

22. Guardar historial de compras de un cliente

Enunciado: Crear un archivo JSON donde cada cliente tenga una lista de compras con producto y cantidad.

```
import json


historial = [
    {"cliente": "Pedro", "compras": [{"producto": "Laptop", "cantidad": 1}]},
    {"cliente": "Lucia", "compras": [{"producto": "Mouse", "cantidad": 2}]}
]


with open("compras.json", "w") as archivo:
    json.dump(historial, archivo, indent=4)


print("Historial de compras guardado")
```

---

23. Agregar una compra a un cliente

Enunciado: Pedir cliente, producto y cantidad y agregarlo al historial de compras.

```
import json

cliente_nombre = input("Ingrese el nombre del cliente: ")
producto = input("Ingrese el producto: ")
cantidad = int(input("Ingrese la cantidad: "))

with open("compras.json", "r") as archivo:
    historial = json.load(archivo)

for registro in historial:
    if registro["cliente"].lower() == cliente_nombre.lower():
        registro["compras"].append({"producto": producto,
"cantidad": cantidad})

with open("compras.json", "w") as archivo:
    json.dump(historial, archivo, indent=4)

print("Compra agregada")
```

---

24. Calcular total de productos comprados por un cliente

Enunciado: Mostrar la cantidad total de productos que ha comprado un cliente.

```
import json

cliente_nombre = input("Ingrese el nombre del cliente: ")

with open("compras.json", "r") as archivo:
    historial = json.load(archivo)

for registro in historial:
    if registro["cliente"].lower() == cliente_nombre.lower():
        total = sum(compra["cantidad"] for compra in
registro["compras"])
        print(f"Total de productos comprados por {cliente_nombre}:
{total}")
```

---

25. Guardar inventario de tienda

Enunciado: Crear un JSON con productos: nombre, precio, stock.

```
import json

inventario = [
    {"nombre": "Teclado", "precio": 50, "stock": 20},
    {"nombre": "Monitor", "precio": 200, "stock": 10}
]

with open("inventario.json", "w") as archivo:
    json.dump(inventario, archivo, indent=4)

print("Inventario guardado")
```

---

26. Actualizar stock de un producto

Enunciado: Modificar la cantidad de un producto en inventario.

```
import json

producto_nombre = input("Ingrese el nombre del producto: ")
nuevo_stock = int(input("Ingrese la nueva cantidad: "))

with open("inventario.json", "r") as archivo:
    inventario = json.load(archivo)

for producto in inventario:
    if producto["nombre"].lower() == producto_nombre.lower():
        producto["stock"] = nuevo_stock

with open("inventario.json", "w") as archivo:
    json.dump(inventario, archivo, indent=4)

print("Stock actualizado")
```

---

27. Guardar configuración de usuario

Enunciado: Crear un archivo JSON con configuraciones de usuario: tema, idioma, notificaciones.

```
import json

config = {"tema": "oscuro", "idioma": "es", "notificaciones":
True}

with open("config.json", "w") as archivo:
    json.dump(config, archivo, indent=4)

print("Configuración guardada")
```

---

28. Cambiar configuración de usuario

Enunciado: Permitir al usuario cambiar una opción de configuración.

```
import json

opcion = input("Ingrese la opción a cambiar
(tema/idioma/notificaciones): ")
valor = input("Ingrese el nuevo valor: ")

with open("config.json", "r") as archivo:
    config = json.load(archivo)

if opcion in config:
    if valor.lower() == "true":
        valor = True
    elif valor.lower() == "false":
        valor = False
    config[opcion] = valor

with open("config.json", "w") as archivo:
    json.dump(config, archivo, indent=4)

print("Configuración actualizada")
```

---

29. Guardar recetas de cocina

Enunciado: Crear un JSON con recetas: nombre, ingredientes, pasos.

```
import json


recetas = [
    {"nombre": "Tortilla", "ingredientes": ["huevos", "patatas"], "pasos": ["pelar patatas", "batir huevos", "freír"]},
    {"nombre": "Ensalada", "ingredientes": ["lechuga", "tomate"], "pasos": ["lavar verduras", "mezclar"]}
]


with open("recetas.json", "w") as archivo:
    json.dump(recetas, archivo, indent=4)


print("Recetas guardadas")
```

---

30. Buscar receta por nombre

Enunciado: Pedir el nombre de una receta y mostrar sus ingredientes y pasos.

```
import json

nombre = input("Ingrese el nombre de la receta: ")

with open("recetas.json", "r") as archivo:
    recetas = json.load(archivo)

encontrada = False
for receta in recetas:
    if receta["nombre"].lower() == nombre.lower():
        print("Ingredientes:", receta["ingredientes"])
        print("Pasos:", receta["pasos"])
        encontrada = True

if not encontrada:
    print("Receta no encontrada")
```

---

31. Guardar agenda de contactos

Enunciado: Crear un JSON con contactos: nombre, teléfono y email.

```
import json


contactos = [
    {"nombre": "Ana", "telefono": "123456789", "email": "ana@mail.com"},
    {"nombre": "Luis", "telefono": "987654321", "email": "luis@mail.com"}
]


with open("contactos.json", "w") as archivo:
    json.dump(contactos, archivo, indent=4)


print("Agenda guardada")
```

---

32. Agregar un contacto nuevo

Enunciado: Pedir al usuario los datos y agregar un nuevo contacto.

```
import json

nombre = input("Nombre: ")
telefono = input("Teléfono: ")
email = input("Email: ")

with open("contactos.json", "r") as archivo:
    contactos = json.load(archivo)

contactos.append({"nombre": nombre, "telefono": telefono, "email": email})

with open("contactos.json", "w") as archivo:
    json.dump(contactos, archivo, indent=4)

print("Contacto agregado")
```

---

33. Buscar un contacto por nombre

Enunciado: Mostrar los datos de un contacto según el nombre.

```
import json

nombre = input("Ingrese el nombre del contacto: ")

with open("contactos.json", "r") as archivo:
    contactos = json.load(archivo)

encontrado = False
for c in contactos:
    if c["nombre"].lower() == nombre.lower():
        print(c)
        encontrado = True

if not encontrado:
    print("Contacto no encontrado")
```

---

34. Eliminar un contacto

Enunciado: Pedir el nombre del contacto y eliminarlo del archivo JSON.

```
import json

nombre = input("Ingrese el nombre del contacto a eliminar: ")

with open("contactos.json", "r") as archivo:
    contactos = json.load(archivo)

contactos = [c for c in contactos if c["nombre"].lower() != nombre.lower()]

with open("contactos.json", "w") as archivo:
    json.dump(contactos, archivo, indent=4)

print("Contacto eliminado si existía")
```

---

35. Guardar biblioteca de libros

Enunciado: Crear un archivo JSON con libros: título, autor y año.

```
import json

libros = [
    {"titulo": "1984", "autor": "George Orwell", "anio": 1949},
    {"titulo": "Cien Años de Soledad", "autor": "Gabriel García Márquez", "anio": 1967}
]

with open("biblioteca.json", "w") as archivo:
    json.dump(libros, archivo, indent=4)

print("Biblioteca guardada")
```

---

36. Buscar libro por título

Enunciado: Mostrar datos de un libro según el título ingresado.

```
import json

titulo = input("Ingrese el título del libro: ")

with open("biblioteca.json", "r") as archivo:
    libros = json.load(archivo)

encontrado = False
for libro in libros:
    if libro["titulo"].lower() == titulo.lower():
        print(libro)
        encontrado = True

if not encontrado:
    print("Libro no encontrado")
```

---

37. Guardar registro de vehículos

Enunciado: Crear un JSON con vehículos: marca, modelo, año y dueño.

```
import json

vehiculos = [
    {"marca": "Toyota", "modelo": "Corolla", "anio": 2015,
"dueno": "Ana"},
    {"marca": "Honda", "modelo": "Civic", "anio": 2018, "dueno":
"Luis"}
]

with open("vehiculos.json", "w") as archivo:
    json.dump(vehiculos, archivo, indent=4)

print("Registro de vehículos guardado")
```

38. Buscar vehículo por dueño

Enunciado: Mostrar todos los vehículos que pertenecen a un dueño.

```
import json

dueno = input("Ingrese el nombre del dueño: ")

with open("vehiculos.json", "r") as archivo:
    vehiculos = json.load(archivo)

encontrados = [v for v in vehiculos if v["dueno"].lower() ==
dueno.lower()]

if encontrados:
    for v in encontrados:
        print(v)
else:
    print("No se encontraron vehículos para ese dueño")
```

---

39. Guardar cuentas bancarias

Enunciado: Crear un JSON con cuentas: titular, número de cuenta y saldo.

```
import json

cuentas = [
    {"titular": "Carlos", "numero": "001", "saldo": 1000},
    {"titular": "Laura", "numero": "002", "saldo": 1500}
]

with open("cuentas.json", "w") as archivo:
    json.dump(cuentas, archivo, indent=4)

print("Cuentas guardadas")
```

---

40. Depositar dinero en una cuenta

Enunciado: Permitir al usuario ingresar dinero en una cuenta según el número de cuenta.

```
import json

numero = input("Ingrese el número de cuenta: ")
monto = float(input("Ingrese el monto a depositar: "))

with open("cuentas.json", "r") as archivo:
    cuentas = json.load(archivo)

for cuenta in cuentas:
    if cuenta["numero"] == numero:
        cuenta["saldo"] += monto

with open("cuentas.json", "w") as archivo:
    json.dump(cuentas, archivo, indent=4)

print("Depósito realizado")
```

---

41. Retirar dinero de una cuenta

Enunciado: Permitir retirar dinero y actualizar el saldo.

```
import json

numero = input("Ingrese el número de cuenta: ")
monto = float(input("Ingrese el monto a retirar: "))

with open("cuentas.json", "r") as archivo:
    cuentas = json.load(archivo)

for cuenta in cuentas:
    if cuenta["numero"] == numero:
        if cuenta["saldo"] >= monto:
            cuenta["saldo"] -= monto
            print("Retiro exitoso")
        else:
            print("Saldo insuficiente")

with open("cuentas.json", "w") as archivo:
    json.dump(cuentas, archivo, indent=4)
```

---

42. Guardar registro de empleados

Enunciado: Crear un JSON con empleados: nombre, puesto, salario.

```
import json

empleados = [
    {"nombre": "Ana", "puesto": "Ingeniera", "salario": 3000},
    {"nombre": "Luis", "puesto": "Diseñador", "salario": 2500}
]

with open("empleados.json", "w") as archivo:
    json.dump(empleados, archivo, indent=4)

print("Empleados guardados")
```

---

43. Aumentar salario de un empleado

Enunciado: Modificar el salario de un empleado según su nombre.

```
import json

nombre = input("Ingrese el nombre del empleado: ")
incremento = float(input("Ingrese el aumento de salario: "))

with open("empleados.json", "r") as archivo:
    empleados = json.load(archivo)

for empleado in empleados:
    if empleado["nombre"].lower() == nombre.lower():
        empleado["salario"] += incremento

with open("empleados.json", "w") as archivo:
    json.dump(empleados, archivo, indent=4)

print("Salario actualizado")
```

44. Guardar seguimiento de ejercicios

Enunciado: Crear un JSON con ejercicios: nombre, repeticiones, fecha.

```
import json


ejercicios = [
    {"nombre": "Flexiones", "repeticiones": 20, "fecha": "2026-03-26"},
    {"nombre": "Sentadillas", "repeticiones": 15, "fecha": "2026-03-26"}
]


with open("ejercicios.json", "w") as archivo:
    json.dump(ejercicios, archivo, indent=4)


print("Registro de ejercicios guardado")
```

---

45. Agregar un nuevo ejercicio

Enunciado: Pedir los datos del ejercicio y agregarlo al archivo.

```
import json

nombre = input("Ejercicio: ")
repeticiones = int(input("Repeticiones: "))
fecha = input("Fecha (YYYY-MM-DD): ")

with open("ejercicios.json", "r") as archivo:
    ejercicios = json.load(archivo)

ejercicios.append({"nombre": nombre, "repeticiones": repeticiones,
"fecha": fecha})

with open("ejercicios.json", "w") as archivo:
    json.dump(ejercicios, archivo, indent=4)

print("Ejercicio agregado")
```

---

46. Guardar menú de restaurante

Enunciado: Crear un JSON con platos: nombre, precio, categoría.

```
import json

menu = [
    {"nombre": "Pizza", "precio": 10, "categoria": "Principal"},
    {"nombre": "Ensalada", "precio": 5, "categoria": "Entrada"}
]

with open("menu.json", "w") as archivo:
    json.dump(menu, archivo, indent=4)

print("Menú guardado")
```

47. Buscar plato por nombre

Enunciado: Mostrar detalles de un plato según el nombre ingresado.

```
import json

nombre = input("Ingrese el nombre del plato: ")

with open("menu.json", "r") as archivo:
    menu = json.load(archivo)

encontrado = False
for plato in menu:
    if plato["nombre"].lower() == nombre.lower():
        print(plato)
        encontrado = True

if not encontrado:
    print("Plato no encontrado")
```

---

48. Actualizar precio de un plato

Enunciado: Cambiar el precio de un plato del menú.

```
import json

nombre = input("Ingrese el nombre del plato: ")
nuevo_precio = float(input("Ingrese el nuevo precio: "))

with open("menu.json", "r") as archivo:
    menu = json.load(archivo)

for plato in menu:
    if plato["nombre"].lower() == nombre.lower():
        plato["precio"] = nuevo_precio

with open("menu.json", "w") as archivo:
    json.dump(menu, archivo, indent=4)

print("Precio actualizado")
```

---

49. Guardar registro de películas

Enunciado: Crear un JSON con películas: título, director, año y género.

```
import json


peliculas = [
    {"titulo": "Inception", "director": "Christopher Nolan", "anio": 2010, "genero": "Ciencia ficción"},
    {"titulo": "Parasite", "director": "Bong Joon-ho", "anio": 2019, "genero": "Thriller"}
]


with open("peliculas.json", "w") as archivo:
    json.dump(peliculas, archivo, indent=4)


print("Películas guardadas")
```

---

50. Buscar película por director

Enunciado: Mostrar todas las películas de un director ingresado.

```
import json


director = input("Ingrese el nombre del director: ")


with open("peliculas.json", "r") as archivo:
    peliculas = json.load(archivo)


encontradas = [p for p in peliculas if p["director"].lower() == director.lower()]


if encontradas:
    for p in encontradas:
        print(p)
else:
    print("No se encontraron películas de ese director")
```

13. LECCIÓN: PROGRAMACIÓN EN PYTHON – FORMATOS DE DATOS

Objetivo de la lección:
Aprender a manejar distintos **formatos de datos en Python**, incluyendo su representación, conversión, lectura y escritura, para poder trabajar con información de manera eficiente y consistente.

---

1. Introducción a los formatos de datos

En programación, un **formato** determina cómo se representa un dato para que pueda ser leído y procesado correctamente.
En Python, los formatos más comunes son:

- **Números**: enteros, decimales, números en notación científica.
- **Cadenas de texto**: caracteres, strings, multilingües.
- **Fechas y horas**: representaciones estándar de fechas, horas y timestamps.
- **Archivos estructurados**: JSON, CSV, XML, YAML, etc.

Importancia: Usar el formato adecuado permite **intercambiar datos entre programas**, **mostrar información legible**, y **procesar datos correctamente**.

---

2. Formato de números en Python

2.1 Enteros y flotantes

- **Enteros (`int`)**: números sin decimales.
- **Flotantes (`float`)**: números con decimales.

Ejemplo:

```
a = 10        # int
b = 3.1416    # float
print(a, b)
```

2.2 Notación científica

Python permite representar números muy grandes o muy pequeños en **notación científica**:

```
x = 1.23e4   # 12300.0
y = 5.67e-3  # 0.00567
print(x, y)
```

2.3 Formateo de números

Para mostrar números con formato:

- `format(valor, ".2f")` → dos decimales

- `f"{valor:.2f}"` → f-strings

Ejemplo:

```
pi = 3.14159265
print(format(pi, ".2f"))       # 3.14
print(f"Pi aproximado: {pi:.3f}")  # Pi aproximado: 3.142
```

---

3. Formato de cadenas de texto

3.1 Tipos de strings

- **Simples:** `'Hola'`
- **Dobles:** `"Mundo"`
- **Multilínea:** `"""Texto largo"""`

3.2 Escapes de caracteres

- `\n` → nueva línea
- `\t` → tabulación
- `\\` → barra invertida
- `\"` → comillas dentro de string

Ejemplo:

```
texto = "Hola\nMundo\tPython"
print(texto)
```

3.3 Concatenación y repetición

- Concatenación: `+`
- Repetición: `*`

Ejemplo:

```
s1 = "Hola"
s2 = "Mundo"
print(s1 + " " + s2)  # Hola Mundo
print(s1 * 3)         # HolaHolaHola
```

3.4 Formato avanzado con f-strings

- **f-strings** permiten incrustar variables y expresiones directamente:

```
nombre = "Ana"
edad = 25
print(f"{nombre} tiene {edad} años")
```

---

4. Conversión de tipos y formatos

Python permite convertir entre tipos para adecuar el formato:

- `int(valor)` → convierte a entero
- `float(valor)` → convierte a decimal
- `str(valor)` → convierte a cadena
- `bool(valor)` → convierte a booleano

Ejemplo:

```
x = "100"
y = int(x) + 50
print(y)  # 150

z = float(x) / 3
print(z)  # 33.3333
```

---

5. Formato de fechas y horas

Python tiene el módulo **`datetime`** para trabajar con fechas y horas.

5.1 Crear fecha y hora

```
from datetime import datetime

ahora = datetime.now()
print(ahora)
```

5.2 Formatear fecha

- `%Y` → año completo
- `%m` → mes con dos dígitos
- `%d` → día con dos dígitos
- `%H` → hora (24h)
- `%M` → minutos
- `%S` → segundos

Ejemplo:

```
fecha = datetime.now()
print(fecha.strftime("%d/%m/%Y %H:%M:%S"))  # 26/03/2026 15:45:30
```

5.3 Convertir string a fecha

```
fecha_str = "2026-03-26 15:45"
fecha_obj = datetime.strptime(fecha_str, "%Y-%m-%d %H:%M")
print(fecha_obj)
```

---

6. Formato de JSON

JSON (JavaScript Object Notation) es un formato de **intercambio de datos estructurado**.

- Se usa con el módulo **`json`** en Python.
- Compatible con **diccionarios, listas, strings, números, booleanos y null**.

Ejemplo de JSON:

```
import json

persona = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}
json_str = json.dumps(persona, indent=4)
print(json_str)
```

- `json.dumps(obj)` → convierte objeto Python a string JSON
- `json.dump(obj, file)` → escribe JSON en archivo
- `json.loads(string)` → convierte string JSON a objeto Python
- `json.load(file)` → lee JSON desde archivo

---

7. Formato CSV

CSV (Comma Separated Values) es un formato **tabular**.
Python usa el módulo **`csv`**:

Escribir CSV:

```
import csv

datos = [["Nombre", "Edad"], ["Ana", 25], ["Luis", 28]]

with open("datos.csv", "w", newline="") as archivo:
    writer = csv.writer(archivo)
    writer.writerows(datos)
```

Leer CSV:

```
import csv

with open("datos.csv", "r") as archivo:
    reader = csv.reader(archivo)
    for fila in reader:
```

```
        print(fila)
```

---

8. Formato de archivos de texto plano

- Se pueden leer y escribir con **`open()`**.
- Modos de apertura:
    - `'r'` → leer
    - `'w'` → escribir (sobrescribe)
    - `'a'` → agregar al final
    - `'x'` → crear archivo nuevo

Ejemplo de escritura:

```
with open("archivo.txt", "w") as f:
    f.write("Hola Mundo\n")
    f.write("Python es genial")
```

Ejemplo de lectura:

```
with open("archivo.txt", "r") as f:
    contenido = f.read()
print(contenido)
```

---

9. Formato binario

- Archivos binarios usan **modo `'rb'` o `'wb'`**
- Útil para imágenes, audio o datos no textuales

Ejemplo:

```
datos = bytes([10, 20, 30, 40])
with open("datos.bin", "wb") as f:
    f.write(datos)

with open("datos.bin", "rb") as f:
    contenido = f.read()
print(contenido)  # b'\n\x14\x1e\x28'
```

---

10. Resumen de formatos y conversiones

Formato	Tipo de datos	Módulo/Función	Uso principal
Números	int, float	`format`, f-strings	Mostrar con decimales

Formato	Tipo de datos	Módulo/Función	Uso principal
Texto	str	f-strings, `str()`	Concatenar, formatear
Fecha/Hora	datetime	`datetime`, `strftime`	Mostrar y parsear fechas
JSON	dict, list	`json`	Intercambio de datos estructurados
CSV	listas, tablas	`csv`	Datos tabulares
Texto plano	str	`open()`	Lectura/escritura de archivos
Binario	bytes	`open('rb/wb')`	Archivos no textuales

---

1. Mostrar número con dos decimales

Enunciado: Mostrar el número 3.14159265 con **dos decimales**.

```
numero = 3.14159265
print(f"Numero formateado: {numero:.2f}")
```

---

2. Mostrar número en notación científica

Enunciado: Mostrar el número 1230000 en **notación científica**.

```
numero = 1230000
print(f"Numero en notacion cientifica: {numero:.2e}")
```

---

3. Alinear texto a la derecha

Enunciado: Imprimir el texto `"Python"` alineado a la **derecha en 20 espacios**.

```
texto = "Python"
print(f"{texto:>20}")
```

---

4. Alinear texto a la izquierda y rellenar con puntos

Enunciado: Mostrar `"Python"` alineado a la **izquierda** y rellenando con `.` hasta 20 caracteres.

```
texto = "Python"
print(f"{texto:.<20}")
```

---

5. Mostrar porcentaje de manera formateada

Enunciado: Mostrar el **porcentaje de 0.856** como `85.6%`.

```
valor = 0.856
print(f"Porcentaje: {valor:.1%}")
```

---

6. Formatear número con separador de miles

Enunciado: Mostrar 1234567 con **coma como separador de miles**.

```
numero = 1234567
print(f"Numero con separador de miles: {numero:,}")
```

---

7. Convertir string a número y sumar

Enunciado: Pedir al usuario un número como **string**, convertirlo a entero y sumarle 10.

```
entrada = input("Ingrese un numero: ")
numero = int(entrada)
suma = numero + 10
print(f"Resultado: {suma}")
```

---

8. Convertir string decimal a flotante y mostrar con 3 decimales

Enunciado: Pedir al usuario un número decimal y mostrarlo con **tres decimales**.

```
entrada = input("Ingrese un numero decimal: ")
numero = float(entrada)
print(f"Numero con 3 decimales: {numero:.3f}")
```

---

9. Mostrar fecha en formato personalizado

Enunciado: Mostrar la fecha actual como `dd/mm/yyyy`.

```
from datetime import datetime
hoy = datetime.now()
print(f"Fecha de hoy: {hoy.strftime('%d/%m/%Y')}")
```

---

10. Mostrar hora en formato 12 horas con AM/PM

Enunciado: Mostrar la hora actual en formato **12 horas con AM/PM**.

```
from datetime import datetime
ahora = datetime.now()
print(f"Hora actual: {ahora.strftime('%I:%M %p')}")
```

---

11. Crear una tabla de multiplicar formateada

Enunciado: Mostrar la tabla de multiplicar del número ingresado por el usuario con **alineación de columnas**.

```
numero = int(input("Ingrese un numero: "))
print(f"Tabla de multiplicar del {numero}")
for i in range(1, 11):
    print(f"{numero} x {i:>2} = {numero*i:>3}")
```

---

12. Alinear lista de productos con precios

Enunciado: Mostrar una lista de productos y precios con **columnas alineadas**.

```
productos = [("Pan", 1.5), ("Leche", 0.99), ("Huevos", 2.75)]
print(f"{'Producto':<10} {'Precio':>6}")
for p, precio in productos:
    print(f"{p:<10} {precio:>6.2f}")
```

---

13. Mostrar números binario, octal y hexadecimal

Enunciado: Convertir un número entero y mostrar sus **formatos binario, octal y hexadecimal**.

```
numero = int(input("Ingrese un numero entero: "))
print(f"Binario: {numero:b}")
print(f"Octal: {numero:o}")
print(f"Hexadecimal: {numero:X}")
```

---

14. Formatear saldo bancario

Enunciado: Mostrar un **saldo monetario** con **dos decimales y separador de miles**.

```
saldo = 1234567.89
print(f"Saldo: ${saldo:,.2f}")
```

---

15. Crear reporte de calificaciones

Enunciado: Mostrar estudiantes y calificaciones en **tabla alineada** con dos decimales.

```
estudiantes = [("Ana", 8.5), ("Luis", 9.25), ("Pedro", 7.75)]
print(f"{'Nombre':<10} {'Calificacion':>12}")
for nombre, cal in estudiantes:
    print(f"{nombre:<10} {cal:>12.2f}")
```

---

16. Convertir string a fecha y mostrar día de la semana

Enunciado: Pedir fecha en formato `YYYY-MM-DD` y mostrar **día de la semana**.

```
from datetime import datetime
fecha_str = input("Ingrese fecha (YYYY-MM-DD): ")
fecha = datetime.strptime(fecha_str, "%Y-%m-%d")
print(f"Día de la semana: {fecha.strftime('%A')}")
```

---

17. Formatear número con signo y espacio

Enunciado: Mostrar números positivos y negativos con **signo explícito y espacio**.

```
numeros = [50, -25, 0]
for n in numeros:
    print(f"{n:+5d}")
```

---

18. Mostrar porcentaje de un examen

Enunciado: Calcular y mostrar el **porcentaje con un decimal**.

```
total = 80
obtenido = 64
porcentaje = obtenido / total
print(f"Porcentaje: {porcentaje:.1%}")
```

---

19. Crear factura simple alineada

Enunciado: Mostrar productos, cantidad, precio unitario y total alineados en **tabla**.

```
productos = [("Pan", 2, 1.5), ("Leche", 1, 0.99)]
print(f"{'Producto':<10} {'Cantidad':>8} {'Precio':>8}
{'Total':>8}")
for nombre, cant, precio in productos:
    total = cant * precio
    print(f"{nombre:<10} {cant:>8} {precio:>8.2f} {total:>8.2f}")
```

---

20. Guardar números en CSV con formato

Enunciado: Guardar una lista de números en **CSV** con **dos decimales**.

```
import csv
numeros = [3.14159, 2.71828, 1.61803]
with open("numeros.csv", "w", newline="") as f:
    writer = csv.writer(f)
    for n in numeros:
        writer.writerow([f"{n:.2f}"])
```

¡Perfecto! Aquí tienes los **programas 21 a 50** de aplicaciones prácticas de Python con **formatos**, nivel principiante a medio. Mantengo la misma estructura: **enunciados numerados en negrita** y partes importantes también en **negrita**.

---

21. Mostrar lista de estudiantes con notas formateadas

Enunciado: Mostrar los nombres y notas de los estudiantes en **tabla con dos decimales**.

```
estudiantes = [("Ana", 8.567), ("Luis", 9.234), ("Pedro", 7.891)]
print(f"{'Nombre':<10} {'Nota':>6}")
for nombre, nota in estudiantes:
    print(f"{nombre:<10} {nota:>6.2f}")
```

---

22. Formatear número como moneda local

Enunciado: Mostrar el número 12345.678 como **moneda con dos decimales y separador de miles**.

```
monto = 12345.678
print(f"Moneda: ${monto:,.2f}")
```

---

23. Crear una tabla de precios alineada

Enunciado: Mostrar productos y precios en **tabla con columnas alineadas a la derecha**.

```
productos = [("Pan", 1.5), ("Leche", 0.99), ("Huevos", 2.75)]
print(f"{'Producto':<10} {'Precio':>8}")
for p, precio in productos:
    print(f"{p:<10} {precio:>8.2f}")
```

---

24. Mostrar número en hexadecimal y binario

Enunciado: Convertir un número entero a **hexadecimal y binario**.

```
numero = int(input("Ingrese un número entero: "))
print(f"Hexadecimal: {numero:X}")
print(f"Binario: {numero:b}")
```

---

25. Convertir string a número decimal y calcular promedio

Enunciado: Pedir tres números como strings, convertirlos a float y calcular el promedio **formateado a dos decimales**.

```
nums = [float(input(f"Ingrese número {i+1}: ")) for i in range(3)]
prom = sum(nums)/3
print(f"Promedio: {prom:.2f}")
```

---

26. Mostrar fecha completa con día de la semana

Enunciado: Mostrar la fecha actual en formato "**Lunes, 26 de Marzo de 2026**".

```
from datetime import datetime
import locale
locale.setlocale(locale.LC_TIME, 'es_ES.UTF-8')
hoy = datetime.now()
print(hoy.strftime("%A, %d de %B de %Y"))
```

---

27. Formatear tiempo en horas, minutos y segundos

Enunciado: Mostrar la hora actual en **HH:MM:SS**.

```
from datetime import datetime
ahora = datetime.now()
print(f"Hora actual: {ahora.strftime('%H:%M:%S')}")
```

---

28. Mostrar alineación de nombres y apellidos

Enunciado: Crear una tabla de nombres y apellidos con **alineación a la izquierda y derecha**.

```
personas = [("Ana", "Gomez"), ("Luis", "Perez"), ("Pedro",
"Lopez")]
print(f"{'Nombre':<10} {'Apellido':>10}")
for nombre, apellido in personas:
    print(f"{nombre:<10} {apellido:>10}")
```

---

29. Crear tabla de inventario con totales

Enunciado: Mostrar productos, cantidad, precio unitario y total alineados.

```
inventario = [("Pan", 10, 1.5), ("Leche", 5, 0.99)]
print(f"{'Producto':<10} {'Cantidad':>8} {'Precio':>8}
{'Total':>8}")
for prod, cant, precio in inventario:
    total = cant * precio
    print(f"{prod:<10} {cant:>8} {precio:>8.2f} {total:>8.2f}")
```

---

30. Guardar números en CSV con formateo

Enunciado: Guardar números flotantes en CSV con **dos decimales**.

```
import csv
numeros = [3.14159, 2.71828, 1.61803]
with open("numeros.csv", "w", newline="") as f:
    writer = csv.writer(f)
    for n in numeros:
        writer.writerow([f"{n:.2f}"])
```

---

31. Crear reporte de empleados

Enunciado: Mostrar nombre, puesto y salario de empleados **alineado en tabla**.

```
empleados = [("Ana", "Ingeniera", 3000), ("Luis", "Diseñador", 2500)]
print(f"{'Nombre':<10} {'Puesto':<12} {'Salario':>8}")
for nombre, puesto, salario in empleados:
    print(f"{nombre:<10} {puesto:<12} {salario:>8.2f}")
```

---

32. Formatear saldo con signo

Enunciado: Mostrar saldos positivos y negativos con **signo explícito**.

```
saldos = [1500, -750, 0]
for s in saldos:
    print(f"{s:+8.2f}")
```

---

33. Mostrar número con relleno de ceros

Enunciado: Mostrar número 42 con **relleno de ceros hasta 5 dígitos**.

```
numero = 42
print(f"{numero:05d}")
```

---

34. Mostrar porcentaje de avance

Enunciado: Mostrar porcentaje de avance de un proyecto con **un decimal**.

```
avance = 0.756
print(f"Avance del proyecto: {avance:.1%}")
```

35. Crear tabla de calificaciones y promedio

Enunciado: Mostrar alumnos, calificación y calcular promedio con **dos decimales**.

```
alumnos = [("Ana", 8.5), ("Luis", 9.25), ("Pedro", 7.75)]
print(f"{'Alumno':<10} {'Calificacion':>12}")
total = 0
for nombre, cal in alumnos:
    total += cal
    print(f"{nombre:<10} {cal:>12.2f}")
promedio = total / len(alumnos)
print(f"{'Promedio':<10} {promedio:>12.2f}")
```

---

36. Guardar productos con precios en CSV

Enunciado: Guardar productos y precios en **CSV con dos decimales**.

```
import csv
productos = [("Pan", 1.5), ("Leche", 0.99), ("Huevos", 2.75)]
with open("productos.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Producto", "Precio"])
    for p, precio in productos:
        writer.writerow([p, f"{precio:.2f}"])
```

---

37. Mostrar fecha y hora completas

Enunciado: Mostrar fecha y hora actuales en **formato DD/MM/YYYY HH:MM:SS**.

```
from datetime import datetime
ahora = datetime.now()
print(ahora.strftime("%d/%m/%Y %H:%M:%S"))
```

---

38. Mostrar fecha en formato largo

Enunciado: Mostrar fecha actual como **“26 de Marzo de 2026”**.

```
from datetime import datetime
import locale
locale.setlocale(locale.LC_TIME, 'es_ES.UTF-8')
hoy = datetime.now()
print(hoy.strftime("%d de %B de %Y"))
```

---

39. Mostrar tabla de inventario con alineación

Enunciado: Alinear productos, cantidad y precio unitario.

```
inventario = [("Pan", 10, 1.5), ("Leche", 5, 0.99)]
print(f"{'Producto':<10} {'Cantidad':>8} {'Precio':>8}")
for p, c, precio in inventario:
    print(f"{p:<10} {c:>8} {precio:>8.2f}")
```

---

40. Mostrar número con formato científico

Enunciado: Mostrar 0.000123 en **notación científica**.

```
numero = 0.000123
print(f"{numero:.3e}")
```

¡Perfecto! Aquí tienes los **programas 41 a 50** de aplicaciones prácticas de Python con **formatos**, completando la colección de 50 programas. Mantengo **enunciados numerados en negrita** y partes importantes también en **negrita**.

---

41. Crear factura detallada

Enunciado: Mostrar productos, cantidad, precio unitario y total con **alineación y separador de miles**.

```
productos = [("Pan", 10, 1.5), ("Leche", 5, 0.99), ("Huevos", 12, 0.25)]

print(f"{'Producto':<10} {'Cantidad':>8} {'Precio':>10} {'Total':>10}")

total_factura = 0

for p, c, precio in productos:

    total = c * precio

    total_factura += total

    print(f"{p:<10} {c:>8} {precio:>10.2f} {total:>10.2f}")

print(f"{'Total Factura':<28} {total_factura:>10.2f}")
```

---

42. Mostrar reporte de empleados con salario y bonificación

Enunciado: Calcular bonificación del 10% y mostrar salario actualizado **formateado con dos decimales**.

```
empleados = [("Ana", 3000), ("Luis", 2500), ("Pedro", 2800)]

print(f"{'Empleado':<10} {'Salario':>8} {'Bonificacion':>12} {'Total':>10}")

for nombre, salario in empleados:

    bono = salario * 0.10

    total = salario + bono

    print(f"{nombre:<10} {salario:>8.2f} {bono:>12.2f} {total:>10.2f}")
```

---

43. Guardar reporte de ventas en CSV con formato

Enunciado: Guardar productos, cantidad y total en **CSV con dos decimales**.

```
import csv
ventas = [("Pan", 10, 1.5), ("Leche", 5, 0.99)]
with open("ventas.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Producto", "Cantidad", "Total"])
    for p, c, precio in ventas:
        total = c * precio
        writer.writerow([p, c, f"{total:.2f}"])
```

---

44. Mostrar números con padding y alineación

Enunciado: Mostrar números enteros de 1 a 10 con **relleno de ceros a 3 dígitos**.

```
for i in range(1, 11):
    print(f"{i:03d}")
```

---

45. Mostrar tabla de notas con promedio y porcentaje

Enunciado: Calcular porcentaje de nota y promedio, mostrar **formato alineado**.

```
alumnos = [("Ana", 85), ("Luis", 92), ("Pedro", 78)]
print(f"{'Alumno':<10} {'Nota':>6} {'Porcentaje':>10}")
total = 0
for nombre, nota in alumnos:
    total += nota
    print(f"{nombre:<10} {nota:>6} {nota/100:.2%}")
promedio = total / len(alumnos)
print(f"{'Promedio':<10} {promedio:>6.2f} {promedio/100:.2%}")
```

---

46. Mostrar fecha y hora en formato personalizado

Enunciado: Mostrar fecha y hora como `26-Mar-2026 03:45 PM`.

```
from datetime import datetime
ahora = datetime.now()
print(ahora.strftime("%d-%b-%Y %I:%M %p"))
```

---

47. Formatear lista de productos con stock bajo

Enunciado: Mostrar productos con stock menor a 5 unidades **alineado en tabla**.

```
productos = [("Pan", 10), ("Leche", 3), ("Huevos", 2)]
print(f"{'Producto':<10} {'Stock':>5}")
for p, s in productos:
    if s < 5:
        print(f"{p:<10} {s:>5}")
```

---

48. Mostrar números flotantes en columnas alineadas

Enunciado: Mostrar una lista de números flotantes **alineados a la derecha con 2 decimales**.

```
numeros = [3.1415, 2.7182, 1.6180, 0.5772]
for n in numeros:
    print(f"{n:>8.2f}")
```

---

49. Mostrar tabla de inventario con total acumulado

Enunciado: Calcular total acumulado de productos y mostrar **tabla alineada**.

```
inventario = [("Pan", 10, 1.5), ("Leche", 5, 0.99), ("Huevos", 12, 0.25)]
print(f"{'Producto':<10} {'Cantidad':>8} {'Precio':>8} {'Total':>8} {'Acumulado':>10}")
acumulado = 0
for p, c, precio in inventario:
    total = c * precio
    acumulado += total
    print(f"{p:<10} {c:>8} {precio:>8.2f} {total:>8.2f} {acumulado:>10.2f}")
```

---

50. Guardar reporte completo de inventario en CSV con totales

Enunciado: Guardar productos, cantidad, precio y total acumulado en **CSV con dos decimales**.

```
import csv

inventario = [("Pan", 10, 1.5), ("Leche", 5, 0.99), ("Huevos", 12, 0.25)]

acumulado = 0

with open("inventario.csv", "w", newline="") as f:

    writer = csv.writer(f)

    writer.writerow(["Producto", "Cantidad", "Precio", "Total", "Acumulado"])

    for p, c, precio in inventario:

        total = c * precio

        acumulado += total

        writer.writerow([p, c, f"{precio:.2f}", f"{total:.2f}", f"{acumulado:.2f}"])
```

---

14. LECCIÓN: PROGRAMACIÓN EN PYTHON – FECHAS Y HORAS

Objetivo de la lección:
Aprender a trabajar con **fechas y horas en Python**, incluyendo creación, formateo, manipulación, comparación y aplicaciones prácticas en programas.

---

1. Introducción a las fechas en Python

En programación, las **fechas y horas** son esenciales para manejar información temporal: registro de eventos, agendas, reportes, cálculos de duración, entre otros.

Python ofrece el módulo **`datetime`** que permite:

- Crear fechas y horas
- Formatearlas en diferentes estilos
- Realizar cálculos entre fechas
- Compararlas

Otros módulos relacionados:

- **`time`** → funciones de tiempo en segundos y temporizadores
- **`calendar`** → trabajar con calendarios y días de la semana

---

2. Importar el módulo datetime

Para usar fechas y horas:

```
import datetime
```

```
from datetime import datetime, date, time, timedelta
```

- `datetime` → objeto que representa **fecha y hora**
- `date` → solo **fecha**
- `time` → solo **hora**
- `timedelta` → diferencias de tiempo

---

3. Obtener la fecha y hora actual

```
ahora = datetime.now()
```

```
print(ahora)
```

- `datetime.now()` → fecha y hora actuales
- `datetime.today()` → fecha y hora actuales (similar)

Ejemplo de salida: `2026-03-26 15:45:30.123456`

---

4. Crear fechas y horas específicas

4.1 Crear una fecha

```
fecha = date(2026, 3, 26)
print(fecha)  # 2026-03-26
```

4.2 Crear una hora

```
hora = time(15, 45, 30)
print(hora)  # 15:45:30
```

4.3 Crear fecha y hora combinadas

```
fecha_hora = datetime(2026, 3, 26, 15, 45, 30)
print(fecha_hora)  # 2026-03-26 15:45:30
```

---

5. Formatear fechas y horas

El método `strftime` permite personalizar la presentación:

Código	Significado
%d	Día del mes con dos dígitos (01-31)
%m	Mes con dos dígitos (01-12)
%Y	Año con cuatro dígitos
%y	Año con dos dígitos
%H	Hora 24h (00-23)
%I	Hora 12h (01-12)
%M	Minutos (00-59)
%S	Segundos (00-59)
%p	AM o PM
%A	Día de la semana completo (Monday)
%a	Día de la semana abreviado (Mon)
%B	Nombre del mes completo (March)
%b	Nombre del mes abreviado (Mar)

Ejemplo:

```
ahora = datetime.now()
print(ahora.strftime("%d/%m/%Y %H:%M:%S"))  # 26/03/2026 15:45:30
print(ahora.strftime("%A, %d de %B de %Y"))  # Thursday, 26 de March de 2026
print(ahora.strftime("%I:%M %p"))  # 03:45 PM
```

---

6. Convertir string a fecha (`strptime`)

```
fecha_str = "26/03/2026 15:45"
fecha_obj = datetime.strptime(fecha_str, "%d/%m/%Y %H:%M")
print(fecha_obj)  # 2026-03-26 15:45:00
```

- `strptime` → **string a objeto datetime**
- Útil para leer fechas desde archivos, formularios o APIs

---

7. Acceder a componentes de fecha y hora

```
ahora = datetime.now()
print("Año:", ahora.year)
print("Mes:", ahora.month)
print("Día:", ahora.day)
print("Hora:", ahora.hour)
print("Minuto:", ahora.minute)
print("Segundo:", ahora.second)
print("Microsegundos:", ahora.microsecond)
```

---

8. Operaciones con fechas (`timedelta`)

- `timedelta` permite **sumar o restar tiempo**:

```
hoy = datetime.now()
mañana = hoy + timedelta(days=1)
ayer = hoy - timedelta(days=1)
print("Hoy:", hoy)
print("Mañana:", mañana)
print("Ayer:", ayer)
```

- También permite horas, minutos, segundos:

```
hoy = datetime.now()
dentro_de_3horas = hoy + timedelta(hours=3)
hace_30min = hoy - timedelta(minutes=30)
print(dentro_de_3horas, hace_30min)
```

---

9. Comparar fechas

```
fecha1 = datetime(2026, 3, 26)
fecha2 = datetime(2026, 4, 1)
if fecha1 < fecha2:
    print("fecha1 es anterior a fecha2")
else:
    print("fecha1 es posterior o igual")
```

- Permite **ordenar listas de fechas** o **verificar vencimientos**

---

10. Diferencia entre fechas

```
inicio = datetime(2026, 3, 20)
fin = datetime(2026, 3, 26)
diferencia = fin - inicio
print("Días de diferencia:", diferencia.days)
```

- `timedelta` también contiene `seconds`, `total_seconds()`

---

11. Calendario y días de la semana

```
import calendar
año = 2026
mes = 3
print(calendar.month(año, mes))  # Muestra calendario del mes
print("Lunes es día número:", calendar.weekday(año, mes, 26))  # 0=Monday
```

---

12. Fechas en múltiples formatos locales

```
import locale
from datetime import datetime
locale.setlocale(locale.LC_TIME, 'es_ES.UTF-8')
hoy = datetime.now()
print(hoy.strftime("%A, %d de %B de %Y"))  # jueves, 26 de marzo de 2026
```

- Cambiar `locale` permite mostrar **meses y días en idioma local**

---

13. Ejemplos prácticos de aplicación de fechas

1. **Calcular edad**

```
fecha_nacimiento = datetime(1990, 5, 10)

hoy = datetime.now()

edad = hoy.year - fecha_nacimiento.year - ((hoy.month, hoy.day) < (fecha_nacimiento.month, fecha_nacimiento.day))

print("Edad:", edad)
```

2. **Contador de días hasta un evento**

```
evento = datetime(2026, 12, 25)

hoy = datetime.now()

dias_restantes = (evento - hoy).days

print("Faltan días para el evento:", dias_restantes)
```

3. **Suma de horas a un horario**

```
hora_inicio = datetime(2026, 3, 26, 14, 30)

hora_fin = hora_inicio + timedelta(hours=2, minutes=45)

print("Hora de finalización:", hora_fin.strftime("%H:%M"))
```

4. **Generar lista de fechas de un mes**

```
from datetime import timedelta

inicio = date(2026, 3, 1)

fechas = [inicio + timedelta(days=i) for i in range(31)]

print(fechas)
```

---

14. Resumen de funcionalidades de fechas en Python

Función / Método	Descripción
`datetime.now()`	Fecha y hora actuales
`date(year, month, day)`	Crear fecha específica
`time(hour, minute, second)`	Crear hora específica
`datetime(year, month, day, hour, minute, second)`	Crear fecha y hora
`strftime(format)`	Convertir fecha a string con formato
`strptime(string, format)`	Convertir string a fecha
`timedelta(days, hours, minutes, seconds)`	Operaciones aritméticas con fechas

Función / Método	Descripción
Comparaciones `<, >, ==`	Comparar fechas
`calendar.month(año, mes)`	Mostrar calendario del mes
`locale.setlocale()`	Mostrar fechas en idioma local

Esta lección cubre:

- Creación y obtención de fechas y horas
- Formateo y conversión entre string y datetime
- Operaciones aritméticas y comparaciones
- Uso de calendarios y localización
- Aplicaciones prácticas: edad, días hasta evento, listas de fechas

---

1. Mostrar fecha y hora actual

Enunciado: Mostrar la **fecha y hora actual**.

```
from datetime import datetime
ahora = datetime.now()
print("Fecha y hora actual:", ahora)
```

---

2. Mostrar solo la fecha actual

Enunciado: Mostrar únicamente la **fecha de hoy**.

```
from datetime import date
hoy = date.today()
print("Fecha de hoy:", hoy)
```

---

3. Mostrar solo la hora actual

Enunciado: Mostrar únicamente la **hora actual**.

```
from datetime import datetime
ahora = datetime.now()
print("Hora actual:", ahora.strftime("%H:%M:%S"))
```

---

4. Crear una fecha específica

Enunciado: Crear una **fecha específica** y mostrarla.

```
from datetime import date
cumple = date(1990, 5, 10)
print("Fecha de cumpleaños:", cumple)
```

---

5. Crear una hora específica

Enunciado: Crear una **hora específica** y mostrarla.

```
from datetime import time
hora_reunion = time(14, 30, 0)
print("Hora de la reunión:", hora_reunion)
```

---

6. Crear fecha y hora combinadas

Enunciado: Crear un **objeto datetime completo**.

```
from datetime import datetime
evento = datetime(2026, 12, 25, 18, 0, 0)
print("Fecha y hora del evento:", evento)
```

---

7. Formatear fecha en estilo dd/mm/yyyy

Enunciado: Mostrar la fecha en **formato dd/mm/yyyy**.

```
from datetime import datetime
ahora = datetime.now()
print("Fecha formateada:", ahora.strftime("%d/%m/%Y"))
```

---

8. Formatear hora en 12 horas con AM/PM

Enunciado: Mostrar la hora en **formato de 12 horas con AM/PM**.

```
from datetime import datetime
ahora = datetime.now()
print("Hora actual 12h:", ahora.strftime("%I:%M %p"))
```

---

9. Convertir string a fecha

Enunciado: Convertir el string `"26/03/2026"` a un **objeto fecha**.

```
from datetime import datetime
fecha_str = "26/03/2026"
fecha_obj = datetime.strptime(fecha_str, "%d/%m/%Y")
print("Objeto fecha:", fecha_obj)
```

---

10. Acceder a componentes de la fecha

Enunciado: Mostrar **año, mes y día** de la fecha actual.

```
from datetime import datetime
hoy = datetime.now()
print("Año:", hoy.year)
print("Mes:", hoy.month)
print("Día:", hoy.day)
```

11. Sumar días a una fecha

Enunciado: Sumar **10 días** a la fecha actual.

```
from datetime import datetime, timedelta
hoy = datetime.now()
dentro_10_dias = hoy + timedelta(days=10)
print("Fecha dentro de 10 días:", dentro_10_dias)
```

---

12. Restar días a una fecha

Enunciado: Restar **5 días** a la fecha actual.

```
from datetime import datetime, timedelta
hoy = datetime.now()
hace_5_dias = hoy - timedelta(days=5)
print("Fecha hace 5 días:", hace_5_dias)
```

---

13. Calcular diferencia entre dos fechas

Enunciado: Calcular la **diferencia en días** entre dos fechas.

```
from datetime import datetime
fecha1 = datetime(2026, 3, 20)
fecha2 = datetime(2026, 3, 26)
diferencia = fecha2 - fecha1
print("Días de diferencia:", diferencia.days)
```

---

14. Comparar dos fechas

Enunciado: Verificar si una fecha es **anterior o posterior** a otra.

```
from datetime import datetime
fecha1 = datetime(2026, 3, 26)
fecha2 = datetime(2026, 4, 1)
if fecha1 < fecha2:
    print("fecha1 es anterior a fecha2")
else:
    print("fecha1 es posterior o igual a fecha2")
```

---

15. Mostrar día de la semana

Enunciado: Mostrar el **día de la semana** de una fecha específica.

```
from datetime import datetime

fecha = datetime(2026, 3, 26)

print("Día de la semana:", fecha.strftime("%A"))
```

---

16. Mostrar mes y nombre completo

Enunciado: Mostrar el **mes en texto completo** de una fecha.

```
from datetime import datetime

fecha = datetime(2026, 3, 26)

print("Mes:", fecha.strftime("%B"))
```

---

17. Crear lista de fechas de un mes

Enunciado: Generar todas las fechas del **mes de marzo de 2026**.

```
from datetime import date, timedelta

inicio = date(2026, 3, 1)

fechas = [inicio + timedelta(days=i) for i in range(31)]

print(fechas)
```

---

18. Calcular edad a partir de la fecha de nacimiento

Enunciado: Calcular la **edad** de una persona.

```
from datetime import datetime

fecha_nacimiento = datetime(1990, 5, 10)

hoy = datetime.now()

edad = hoy.year - fecha_nacimiento.year - ((hoy.month, hoy.day) < (fecha_nacimiento.month, fecha_nacimiento.day))

print("Edad:", edad)
```

---

19. Contar días hasta un evento

Enunciado: Calcular los **días restantes** hasta el 25 de diciembre de 2026.

```
from datetime import datetime
evento = datetime(2026, 12, 25)
hoy = datetime.now()
dias_restantes = (evento - hoy).days
print("Días hasta el evento:", dias_restantes)
```

---

20. Sumar horas a un horario

Enunciado: Calcular la **hora de finalización** sumando 2 horas y 30 minutos.

```
from datetime import datetime, timedelta
hora_inicio = datetime(2026, 3, 26, 14, 30)
hora_fin = hora_inicio + timedelta(hours=2, minutes=30)
print("Hora de finalización:", hora_fin.strftime("%H:%M"))
```

---

21. Mostrar calendario de un mes

Enunciado: Mostrar el **calendario completo de marzo de 2026**.

```
import calendar
año = 2026
mes = 3
print(calendar.month(año, mes))
```

---

22. Obtener número del día de la semana

Enunciado: Mostrar el **número del día de la semana** de una fecha (0=Lunes).

```
import calendar
from datetime import date
fecha = date(2026, 3, 26)
numero_dia = calendar.weekday(fecha.year, fecha.month, fecha.day)
print("Número del día de la semana (0=Lunes):", numero_dia)
```

---

23. Mostrar días restantes de la semana

Enunciado: Calcular cuántos **días faltan para el domingo**.

```
from datetime import datetime, timedelta
hoy = datetime.now()
dias_hasta_domingo = 6 - hoy.weekday()  # 0=Lunes, 6=Domingo
print("Días hasta el domingo:", dias_hasta_domingo)
```

---

24. Sumar semanas a una fecha

Enunciado: Sumar **3 semanas** a la fecha actual.

```
from datetime import datetime, timedelta
hoy = datetime.now()
dentro_3_semanas = hoy + timedelta(weeks=3)
print("Fecha dentro de 3 semanas:", dentro_3_semanas)
```

---

25. Restar meses a una fecha (aproximado)

Enunciado: Restar **2 meses** a la fecha actual (30 días por mes).

```
from datetime import datetime, timedelta
hoy = datetime.now()
hace_2_meses = hoy - timedelta(days=60)
print("Fecha hace 2 meses:", hace_2_meses)
```

---

26. Calcular días hábiles entre dos fechas

Enunciado: Contar los **días laborables** entre dos fechas.

```
from datetime import date, timedelta
inicio = date(2026, 3, 1)
fin = date(2026, 3, 10)
dias_habiles = sum(1 for i in range((fin - inicio).days + 1) if
(inicio + timedelta(days=i)).weekday() < 5)
print("Días hábiles:", dias_habiles)
```

---

27. Mostrar fecha en formato largo local

Enunciado: Mostrar fecha con **día y mes en español**.

```
from datetime import datetime
import locale
locale.setlocale(locale.LC_TIME, 'es_ES.UTF-8')
hoy = datetime.now()
print(hoy.strftime("%A, %d de %B de %Y"))
```

---

28. Calcular edad exacta con meses y días

Enunciado: Calcular **años, meses y días** desde la fecha de nacimiento.

```
from datetime import datetime
fecha_nacimiento = datetime(1990, 5, 10)
hoy = datetime.now()
anos = hoy.year - fecha_nacimiento.year - ((hoy.month, hoy.day) < (fecha_nacimiento.month, fecha_nacimiento.day))
meses = (hoy.month - fecha_nacimiento.month - (hoy.day < fecha_nacimiento.day)) % 12
dias = (hoy - datetime(hoy.year, hoy.month, fecha_nacimiento.day)).days % 30
print(f"Edad: {anos} años, {meses} meses, {dias} días")
```

---

29. Verificar si un año es bisiesto

Enunciado: Determinar si un año es **bisiesto**.

```
import calendar
año = 2026
if calendar.isleap(año):
    print(año, "es bisiesto")
else:
    print(año, "no es bisiesto")
```

---

30. Mostrar la fecha y hora en formato ISO

Enunciado: Mostrar la fecha en **formato ISO 8601**.

```
from datetime import datetime

ahora = datetime.now()

print("Formato ISO:", ahora.isoformat())
```

---

31. Calcular fin de semana siguiente

Enunciado: Obtener la fecha del **próximo sábado** a partir de hoy.

```
from datetime import datetime, timedelta

hoy = datetime.now()

dias_hasta_sabado = (5 - hoy.weekday()) % 7

proximo_sabado = hoy + timedelta(days=dias_hasta_sabado)

print("Próximo sábado:", proximo_sabado.date())
```

---

32. Crear lista de fechas de un rango

Enunciado: Generar todas las **fechas entre dos fechas dadas**.

```
from datetime import date, timedelta

inicio = date(2026, 3, 20)

fin = date(2026, 3, 26)

fechas = [inicio + timedelta(days=i) for i in range((fin - inicio).days + 1)]

print(fechas)
```

---

33. Mostrar días de la semana de un mes

Enunciado: Mostrar los **días de la semana** de marzo de 2026.

```
import calendar

año = 2026

mes = 3

for dia in range(1, 32):

    dia_semana = calendar.day_name[calendar.weekday(año, mes, dia)]

    print(f"{dia}/{mes}/{año} - {dia_semana}")
```

---

34. Contar días hasta fin de año

Enunciado: Calcular los **días restantes hasta el 31 de diciembre**.

```
from datetime import datetime
hoy = datetime.now()
fin_año = datetime(hoy.year, 12, 31)
dias_restantes = (fin_año - hoy).days
print("Días hasta fin de año:", dias_restantes)
```

---

35. Agregar minutos a una hora

Enunciado: Sumar **90 minutos** a la hora actual.

```
from datetime import datetime, timedelta
ahora = datetime.now()
nueva_hora = ahora + timedelta(minutes=90)
print("Hora +90 minutos:", nueva_hora.strftime("%H:%M"))
```

---

36. Restar horas a una hora

Enunciado: Restar **3 horas** a la hora actual.

```
from datetime import datetime, timedelta
ahora = datetime.now()
hora_anterior = ahora - timedelta(hours=3)
print("Hora -3 horas:", hora_anterior.strftime("%H:%M"))
```

---

37. Generar fechas de lunes a viernes

Enunciado: Crear lista con todas las **fechas hábiles de marzo de 2026**.

```
from datetime import date, timedelta
inicio = date(2026, 3, 1)
fin = date(2026, 3, 31)
fechas_habiles = [inicio + timedelta(days=i) for i in range((fin - inicio).days + 1) if (inicio + timedelta(days=i)).weekday() < 5]
print(fechas_habiles)
```

---

38. Calcular edad en años y meses

Enunciado: Calcular la **edad aproximada en años y meses**.

```
from datetime import datetime

fecha_nac = datetime(1990, 5, 10)

hoy = datetime.now()

anos = hoy.year - fecha_nac.year - ((hoy.month, hoy.day) < (fecha_nac.month, fecha_nac.day))

meses = (hoy.month - fecha_nac.month - (hoy.day < fecha_nac.day)) % 12

print(f"Edad: {anos} años, {meses} meses")
```

---

39. Mostrar fecha con formato personalizado

Enunciado: Mostrar fecha como "**jueves, 26 de marzo de 2026**".

```
from datetime import datetime

import locale

locale.setlocale(locale.LC_TIME, 'es_ES.UTF-8')

hoy = datetime.now()

print(hoy.strftime("%A, %d de %B de %Y"))
```

---

40. Guardar fecha y hora en archivo

Enunciado: Guardar la **fecha y hora actual** en un archivo `fecha.txt`.

```
from datetime import datetime

ahora = datetime.now()

with open("fecha.txt", "w") as f:

    f.write("Fecha y hora actual: " + ahora.strftime("%d/%m/%Y %H:%M:%S"))

print("Fecha guardada en fecha.txt")
```

¡Perfecto! Aquí tienes los **programas 41 a 50** de aplicaciones prácticas de Python con **fechas y horas**, completando la colección de 50 programas. Mantengo **enunciados en negrita**, partes importantes también en **negrita**, tamaño de fuente 12 implícito.

---

41. Crear recordatorio para un evento

Enunciado: Mostrar un mensaje si un evento ocurre **hoy**.

```
from datetime import datetime
evento = datetime(2026, 3, 26)
hoy = datetime.now()
if hoy.date() == evento.date():
    print("¡El evento es hoy!")
else:
    print("El evento no es hoy.")
```

---

42. Calcular días vencidos

Enunciado: Calcular cuántos **días han pasado desde una fecha de vencimiento**.

```
from datetime import datetime
vencimiento = datetime(2026, 3, 20)
hoy = datetime.now()
dias_vencidos = (hoy - vencimiento).days
print("Días vencidos:", dias_vencidos)
```

---

43. Generar agenda semanal

Enunciado: Crear lista con **fechas de lunes a domingo** de la semana actual.

```
from datetime import datetime, timedelta
hoy = datetime.now()
inicio_semana = hoy - timedelta(days=hoy.weekday())
agenda = [inicio_semana + timedelta(days=i) for i in range(7)]
print("Agenda semanal:", agenda)
```

---

44. Guardar fechas de eventos en CSV

Enunciado: Guardar **eventos y fechas** en un archivo CSV.

```
import csv
from datetime import datetime
eventos = [("Cumpleaños Ana", datetime(2026, 5, 10)),
           ("Reunión", datetime(2026, 3, 26)),
           ("Navidad", datetime(2026, 12, 25))]
with open("eventos.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Evento", "Fecha"])
    for e, fecha in eventos:
        writer.writerow([e, fecha.strftime("%d/%m/%Y")])
print("Eventos guardados en eventos.csv")
```

---

45. Leer fechas de un archivo CSV

Enunciado: Leer las **fechas de eventos** desde un CSV.

```
import csv
from datetime import datetime
with open("eventos.csv", "r") as f:
    reader = csv.DictReader(f)
    for row in reader:
        evento = row["Evento"]
        fecha = datetime.strptime(row["Fecha"], "%d/%m/%Y")
        print(evento, "-", fecha.date())
```

---

46. Calcular fecha de pago

Enunciado: Calcular la **fecha de pago** 15 días después del día de hoy.

```
from datetime import datetime, timedelta
hoy = datetime.now()
fecha_pago = hoy + timedelta(days=15)
print("Fecha de pago:", fecha_pago.strftime("%d/%m/%Y"))
```

---

47. Mostrar horario de reuniones con formato

Enunciado: Mostrar la **hora de inicio y fin** de varias reuniones.

```
from datetime import datetime, timedelta

reuniones = [("Reunión 1", datetime(2026, 3, 26, 9, 0), 60),
             ("Reunión 2", datetime(2026, 3, 26, 11, 0), 90)]

for nombre, inicio, duracion_min in reuniones:
    fin = inicio + timedelta(minutes=duracion_min)
    print(f"{nombre}: {inicio.strftime('%H:%M')} -
{fin.strftime('%H:%M')}")
```

---

48. Contador de días hasta evento recurrente

Enunciado: Calcular cuántos **días faltan para el próximo lunes**.

```
from datetime import datetime, timedelta

hoy = datetime.now()

dias_hasta_lunes = (0 - hoy.weekday()) % 7

proximo_lunes = hoy + timedelta(days=dias_hasta_lunes)

print("Días hasta el próximo lunes:", dias_hasta_lunes, "Fecha:",
proximo_lunes.date())
```

---

49. Mostrar fecha de inicio y fin de mes

Enunciado: Obtener la **primera y última fecha del mes actual**.

```
from datetime import datetime

import calendar

hoy = datetime.now()

primer_dia = datetime(hoy.year, hoy.month, 1)

ultimo_dia = datetime(hoy.year, hoy.month,
calendar.monthrange(hoy.year, hoy.month)[1])

print("Primer día del mes:", primer_dia.date())

print("Último día del mes:", ultimo_dia.date())
```

---

50. Guardar reporte de fechas en JSON

Enunciado: Guardar varias **fechas importantes** en un archivo JSON.

```
import json
from datetime import datetime
fechas_importantes = {
    "Cumpleaños Ana": datetime(2026, 5, 10).strftime("%d/%m/%Y"),
    "Navidad": datetime(2026, 12, 25).strftime("%d/%m/%Y"),
    "Reunión Proyecto": datetime(2026, 3, 26).strftime("%d/%m/%Y")
}
with open("fechas.json", "w") as f:
    json.dump(fechas_importantes, f, indent=4)
print("Fechas importantes guardadas en fechas.json")
```

Epílogo

Llegar al final de este libro significa más que haber leído un conjunto de conceptos y ejercicios; significa haber dado un paso real hacia convertirte en un programador capaz de enfrentar problemas y crear soluciones con **Python**. La programación es un camino de aprendizaje continuo: cada línea de código que escribes, cada error que corriges y cada proyecto que completas te acercan a dominar esta poderosa herramienta.

Espero que este libro te haya mostrado que **aprender practicando** es la forma más efectiva de internalizar conocimientos y desarrollar confianza. La teoría es importante, pero la acción transforma ideas en habilidades. Los retos, los experimentos y los proyectos que has realizado aquí son solo el comienzo de un viaje que puede llevarte a descubrir nuevas tecnologías, resolver problemas complejos y crear soluciones que impacten en el mundo real.

Quiero cerrar estas páginas nuevamente con un agradecimiento especial a **Marisa**, cuya paciencia, apoyo y entusiasmo fueron un pilar fundamental durante la creación de este libro. Su confianza en mí hizo posible que este proyecto se concretara y que cada capítulo fuera escrito con dedicación y pasión.

Ahora, querido lector, el próximo paso está en tus manos. Sigue practicando, explorando y creando. Python es solo una herramienta: tu curiosidad y perseverancia son las que realmente harán la diferencia. ¡Que tu aprendizaje nunca se detenga y que tu código siempre abra nuevas posibilidades!

Luis Minguillón Pascual